Controlling Rigid Bodies with Dynamic Constraints

Ronen Barzel

Computer Science Department
California Institute of Technology

Caltech-CS-TR-88-19

Controlling Rigid Bodies with Dynamic Constraints

Ronen Barzel
California Institute of Technology
Pasadena, CA 91125

November 16, 1988

Caltech-CS-TR-88-19

Abstract

We present a technique, "Dynamic Constraints," for controlling the positions and orientations of rigid bodies in computer graphics models. The technique addresses the issue of providing control over a *physically-based* model, in which the bodies' behavior is determined by simulation of Newton's Laws of Motion. The "Dynamic Constraints" technique allows the modeler to control the configuration of bodies in a model by specifying constraints which must be met and maintained, as the simulation unfolds. To meet the constraints, we introduce forces into the model; these "constraint forces" forces are applied to the bodies that are constrained. We use an *inverse dynamics* method to derive a linear "constraint-force equation," whose solution yields the values of the constraint forces. We continually solve the constraint-force equation during the simulation of a model, so that the bodies' behavior will be consistent with the constraints. This thesis is accompanied by a narrated 10-minuted videotape, "Caltech Modeling Demos, 1987" that contains several animations demonstrating the "Dynamic Constraints" technique. This thesis additionally includes a glossary of terms relating to physically-based modeling and dynamic constraints.

Acknowledgements

My advisor, Al Barr, had the seminal idea for constraint-based control of dynamic rigid bodies; we have worked jointly in developing the "Dynamic Constraints" technique, and the inverse dynamics method it is based on. Some people who additionally deserve mention: Jed Lengyel, for being a "guinea pig" user of the modeling system, and for putting together the video system that was used in making animations; John Snyder, for the rendering of the animations; John Platt, for discussions and for numerical software; and Brian Von Herzen, for the motivation and modeling for the space-station self-assembly animations. Supporters of the graphics lab, who made available equipment that was used in this work: Ampex, a VPR-3 video tape recorder; Hewlett-Packard, several HP9000 machines that were used for rendering; Schlumberger Palo Alto Research Center and Symbolics, Inc., Symbolics Lisp Machines that were used to implement the Dynamic Constraints technique. Finally, I have been personally supported, initially by Caltech, and now by the AT&T Foundation, as the recipient of an AT&T Bell Laboratories Ph.D. Scholarship.

Contents

Index of Figures

List of Symbols

Conventions:

- A Super-arrow indicates a vector; e.g. $\vec{b}$.
- Boldface indicates a matrix; e.g. $\mathbf{\Gamma}$.
- The superscript i is used as a a label for a body; e.g. $\vec{F}^i$ is the net force on the ith body. If only one body is being discussed, the superscript is sometimes left off.
- The subscripts j and k are typically used as labels for a constraint; e.g. $\vec{D}_k$ is the deviation function for kth constraint. If only one constraint is being discussed, the subscript is sometimes left off.
- The subscript *body* indicates a quantity that is represented in the body coordinate system of a body; e.g. $\mathbf{I}_{body}$.
- Other subscripts are part of the "name" of an object, e.g. $\vec{F}$, the net force on a body, vs. $\vec{F}_N$, the net non-constraint force on a body.
- Calligraphic font is used for single quantities that contain values for the entire model; e.g. $\mathcal{F}$ is the collection of forces acting on bodies in the model.

Greek symbols:

$\vec{\beta}_k$ The part of $\vec{D}_k^{(2)}(\mathcal{Y},\mathcal{F},\mathcal{T},t)$ independent of force and torque; a d_k-component vector. [Fig. 3.6]

$\vec{\beta}$ $\vec{\beta}_k$ with the subscript dropped, when there is only one constraint. [Eqn. 3.6]

$\mathbf{\Gamma}_k^i$ A matrix that acts on the net force on body i, yielding a contribution to $\vec{D}_k^{(2)}(\mathcal{Y},\mathcal{F},\mathcal{T},t)$. [Fig. 3.6]

$\mathbf{\Gamma}^i$ $\mathbf{\Gamma}_k^i$ with the subscript dropped, when there is only one constraint. [Eqn. 3.6]

$\mathbf{\Gamma}$ $\mathbf{\Gamma}^i$ with the superscript i dropped, for a model with only a single body.

$\mathbf{\Lambda}_k^i$ A matrix that acts on the net torque on body i, yielding a contribution to $\vec{D}_k^{(2)}(\mathcal{Y},\mathcal{F},\mathcal{T},t)$. [Fig. 3.6]

$\mathbf{\Lambda}^i$ $\mathbf{\Lambda}_k^i$ with the subscript dropped, when there is only one constraint. [Eqn. 3.6]

$\mathbf{\Lambda}$ $\mathbf{\Lambda}^i$ with the superscript i dropped, for a model with only a single body.

τ The time-constant for the critically-damped decay of D; determines the rate of assembly of a constraint. [Eqn. 3.4]

τ_k The time-constant τ for constraint k. [Eqn. 3.17]

Roman symbols:

$\vec{a}$ Acceleration. [Appendix B.3]

$\mathcal{B}$ The constant term of the constraint-force equation $\mathcal{M}\mathcal{F}_c = \mathcal{B}$; a multidimensional vector. [Eqn. 3.18]

$\mathcal{B}_k$ The kth element of $\mathcal{B}$; corresponds to the kth constraint. [Eqn. 3.18, Fig. 3.6]

$\vec{b}$ Radius vector from center-of-mass of a body to a point on the body. [Appendix B.3]

$\vec{b}_{body}$ $\vec{b}$ expressed in body coordinates; constant for a point fixed on a body. [Appendix B.3]

$\mathbf{b}^*$ The dual matrix to $\vec{b}$; for all $\vec{a}$, $\mathbf{b}^*\vec{a} \equiv \vec{b} \times \vec{a}$. [Appendix B.2]

$\mathbf{b}^{*\mathrm{T}}$ The transpose of $\mathbf{b}^*$; $\mathbf{b}^{*\mathrm{T}} = -\mathbf{b}^*$. [Appendix B.2]

d The number of components of $\vec{D}$ for a constraint. [Eqn. 3.6]

d_k The number of components of $\vec{D}$ for constraint k. [Fig. 3.6]

$\vec{D}(\mathcal{Y}_s,t)$ The deviation function of a constraint; a d-component measure of the spatial state of the model. [Eqn. 3.1]

$\vec{D}(\mathcal{Y}_s)$ $\vec{D}(\mathcal{Y}_s,t)$ with the time parameter dropped, for constraints that don't have explicit dependence on time.

$\vec{D}, \vec{D}()$ $\vec{D}(\mathcal{Y}_s,t)$ with the parameters dropped for notational convenience.

$\vec{D}_0$ The initial value of $\vec{D}$ when a constraint is turned on. [Eqn. 3.3]

$\vec{D}_k$ $\vec{D}$ for constraint k. [Fig. 3.6]

$\vec{D}^{(1)}(\mathcal{Y},t)$ A function of the state of the model that measures the velocity of $\vec{D}$; $\vec{D}^{(1)}(\mathcal{Y}(t),t) \equiv \frac{d}{dt}\vec{D}(\mathcal{Y}_s(t),t)$. [Eqn. 3.5]

$\vec{D}^{(1)}$ $\vec{D}^{(1)}(\mathcal{Y},t)$ with the parameters dropped for notational convenience.

$\vec{D}_k^{(1)}$ $\vec{D}^{(1)}$ for constraint k. [Fig. 3.6]

$\vec{D}^{(2)}(\mathcal{Y},\mathcal{F},\mathcal{T},t)$ A function of the state of the model that measures the acceleration of $\vec{D}$; $\vec{D}^{(2)}(\mathcal{Y}(t),\mathcal{F}(t),\mathcal{T}(t),t) \equiv \frac{d^2}{dt^2}\vec{D}(\mathcal{Y}_s(t),t)$. [Eqn. 3.5]

$\vec{D}^{(2)}$ $\vec{D}^{(2)}(\mathcal{Y},\mathcal{F},\mathcal{T},t)$ with the parameters dropped for notational convenience.

$\vec{D}_k^{(2)}$ $\vec{D}^{(2)}$ for constraint k. [Fig. 3.6]

$\mathcal{F}$ The collection of forces acting on bodies in the model. [Eqn. A.5]

f The number of components of $\vec{F}_c$ for a constraint. [Eqn. 3.9]

f_j The number of components of $\vec{F}_c$ for constraint j. [Eqn. 3.9]

$\mathcal{F}_c$ The collection of "constraint forces" $\vec{F}_c$ of all the constraints in the model; a multidimensional vector whose value is computed by solving the constraint-force equation $\mathcal{M}\mathcal{F}_c = \mathcal{B}$. [Eqn. 3.18, Fig. 3.6]

$\mathcal{F}_{cj}$ The jth element of $\mathcal{F}_c$; the constraint force for the jth constraint. [Eqn. 3.18, Fig. 3.6]

$\vec{F}_c$ The generalized force for a constraint, an f-component vector from which the constraint force and constraint torque are computed; $\vec{F}_c$ itself is loosely called the "constraint force." [Sec. 3.3.2]

$\vec{F}_{cj}$ $\vec{F}_c$ for constraint j. [Eqn. 3.9, Fig. 3.6]

$\vec{F}$ The net force on a body. [Appendix A.1]

$\vec{F}^i$ The net force on body i. [Eqn. 3.6]

$\vec{F}_N$ The net non-constraint force on a body. [Eqn. 3.8]

$\vec{F}_N^i$ The net non-constraint force on body i. [Fig. 3.6]

$\mathbf{G}_j^i$ A $3 \times f_j$ matrix determining the force on body i due to constraint j. [Fig. 3.6]

$\mathbf{G}$ $\mathbf{G}_j^i$ with the i and j dropped, for a single body with a single constraint. [Eqn. 3.9]

$\mathbf{H}_j^i$ A $3 \times f_j$ matrix determining the torque on body i due to constraint j. [Fig. 3.6]

$\mathbf{H}$ $\mathbf{H}_j^i$ with the i and j dropped, for a single body with a single constraint. [Eqn. 3.9]

$\mathbf{I}$ The rotational inertia tensor for a body. [Appendix A.1]

$\mathbf{I}^{-1}$ The inverse of $\mathbf{I}$. [Appendix A.1]

$\mathbf{I}_{body}^{-1}$ The inverse of $\mathbf{I}$, expressed in body coordinates; constant for a rigid body. [Appendix A.1]

$\vec{L}$ The angular momentum of a body. [Appendix A.1]

m The mass of a body. [Appendix A.1]

$\mathcal{M}$ The matrix term of the constraint-force equation $\mathcal{M}\mathcal{F}_c = \mathcal{B}$. [Eqn. 3.18]

$\mathcal{M}_{kj}$ The $[k,j]$th element of $\mathcal{M}$. [Eqn. 3.18, Fig. 3.6]

$\vec{p}$ The linear momentum of a body. [Appendix A.1]

$\mathbf{Q}$ The quaternion representation of the orientation of a body. [Appendix A.1]

$\mathbf{R}$ The orientation matrix for a body. [Appendix A.1]

$\mathbf{R}^T$ The transpose of $\mathbf{R}$.

t Time; the independent parameter of the simulation.

t_0 The time at which a constraint is introduced. [Eqn. 3.3]

$\vec{T}$ The net torque on a body. [Appendix A.1]

$\vec{T}^i$ The net torque on body i. [Eqn. 3.6]

$\vec{T}_N$ The net non-constraint torque on a body. [Eqn. 3.8]

$\vec{T}_N^i$ The net non-constraint torque on body i. [Fig. 3.6]

$\mathcal{T}$ The collection of torques acting on bodies in the model. [Eqn. A.5]

$\vec{v}$ Velocity of a point or body. [Appendix A.1, Appendix B.3]

$\vec{\omega}$ The angular velocity of a body. [Appendix A.1]

ω^* The dual matrix to $\vec{\omega}$; for all $\vec{a}$, $\omega^*\vec{a} \equiv \vec{\omega} \times \vec{a}$. [Appendix B.2]

ω^{*T} The transpose of ω^*; $\omega^{*T} = -\omega^*$. [Appendix B.2]

$\vec{x}$ The position of the center of mass of a body. [Appendix A.1]

$\vec{X}_0$ The position of a "nail"; a constant vector. [Eqn. 3.2,Fig. 3.2]

$\vec{X}_P$ The position of a point P on a body; a function of the state of the body. [Appendix B.3]

$\vec{X}^{\text{path}}$ A position along a path; a function of time. [Appendix C.3.2, Fig. C.1]

$\mathcal{Y}$ The collection of the states of the bodies in the model; loosely, $\mathcal{Y}$ is referred to as the state of the model. [Eqn. A.5]

$\mathcal{Y}_s$ The spatial state of the model, i.e. the collection of positions and orientations of the bodies. [Eqn. 3.1]

Glossary

angular momentum As in Newtonian mechanics, symbolized by $\vec{L}$. See Appendix A.1.

angular velocity As in Newtonian mechanics, symbolized by $\vec{\omega}$. See Appendix A.1.

animation Successive images are rendered, producing the frames of a motion picture. The "traditional" animation techniques (See *traditional modeling.*) are kinematic; after building the models of objects, the animator adjusts positioning parameters as a function of frame number. For physically-based modeling, the temporal behavior of objects is bound up in the model; thus there is less of a distinction between modeling and animation.

auxiliary variable A physical quantity associated with a body, that can be computed from the body's *state variables* (q.v.). For example, a body's *velocity* $\vec{v}$ is computed from its state variables momentum $\vec{p}$ and mass m via $\vec{v} = \frac{1}{m}\vec{p}$. See Appendix A.

body coordinates Each primitive body is typically described in its own coordinate system; for example, a rod may be described as being a certain length along the z-axis, and a certain radius in the x-y plane. If the body is in an arbitrary orientation in space, vectors in body coordinates can be converted into world coordinates (q.v.) via the body's orientation matrix (q.v.).

center-of-mass As in Newtonian mechanics. The position of the center of mass of a body is symbolized by $\vec{x}$. See Appendix A.1.

classical dynamics The well-known principles of motion of physical bodies; these are based on Newton's laws and Euler's principle (q.v.). See also: *dynamics.*.

compound object An object built from a collection of other objects. The ultimate building blocks are the *primitive bodies* (q.v.). See also: *self-assembly.*

constraint A restriction on the "legal" state of a model. The "Dynamic Constraints" technique focuses on *geometric constraints*, which are constraints on the positions and orientations of the bodies in a model. See Sec. 1.3.

constraint force A force that is applied to a constrained body to cause it to move such that the constraint is met. See Sec. 2.1. Also, the term "constraint force" is loosely applied to the quantity $\vec{F}_c$ associated with a constraint, from which the actual constraint forces and torques are calculated. See Sec. 3.3.2.

constraint-force equation The linear equation $\mathcal{M}\mathcal{F}_c + \mathcal{B} = 0$, whose solution $\mathcal{F}_c$ yields the constraint forces that meet the collection of constraints in a model. See derivation in Ch. 3. As we simulate the behavior of the bodies in a model, we continually set up and solve this equation, so that the constraint forces adapt to the state of the model. See Sec. 4.2.

control-point In the modeling system (Ch. 5), a point to which forces or constraints can be attached. Two common types of control-points are those that are fixed on a body (see Fig. A.2), and those that are fixed in space. The latter are called *nails.* See Appendix 5.2.

deviation function For a given constraint, the deviation function $\vec{D}(\mathcal{Y}_s, t)$ measures how much the state of the model differs from a state in which the constraint is met: $\vec{D}(\mathcal{Y}_s, t) = 0$ if and only if the constraint is met with the bodies in state $\mathcal{Y}_s$ at time t. See Sec. 3.1.

differential-equation solver A subroutine that numerically solves a differential equation. The differential-equation solver is the basic tool used for simulating classical dynamics. See Appendix A.3.3 and Appendix A.3.4.

dual The dual of a vector $\vec{b}$ is the matrix $\mathbf{b}^*$ such that for any vector $\vec{a}$, $\mathbf{b}^*\vec{a} \equiv \vec{b} \times \vec{a}$. See Appendix B.2.

dynamic Relating to motion taking into account effects of force and inertia. See also: *kinematic.*

dynamic constraints The computer graphics modeling technique for meeting constraints in a physically-based model by applying constraint forces that are computed via an inverse-dynamics technique.

einstein summation notation (ESN) A mathematical notation useful for tensor and linear-algebra calculations: matrix and vector components are labeled by subscripts; a subscript used twice in a term implies a summation. Thus the ESN expression $x_i = A_{ij}B_{kj}y_k$ is equivalent to the linear algebraic expression $\vec{x} = \mathbf{A}\mathbf{B}^T\vec{y}$. See [Barr 83], [Misner, Thorne and Wheeler 73].

Euler's principle of superposition The principle that allows the summing of all forces on a body to yield a net force (q.v.) and net torque (q.v.) that act on the center of mass; also allows pure torques to act on the body and contribute to the net torque.

external applied force (torque) In the modeling system, a force (torque) that the modeler explicitly applies to the bodies in the model; e.g. gravity. See Sec. 5.1. See also: *non-constraint force (torque).*

forward dynamics The problem of computing the motion of a collection of bodies, given the forces and torques acting on them; to solve this is a straightforward application of the laws of classical dynamics. See also: *inverse dynamics.*

forward kinematics The problem of computing the configuration of a collection of bodies, given the positioning transformations acting on them; to solve this is a straightforward matter of performing the appropriate linear algebra. See also: *inverse kinematics.*

geometric constraint See *constraint.*

internal force The internal forces of a compound object (q.v.) are the forces that hold the component bodies together. For example, Fig. 2.1 shows the internal forces of a compound pendulum. See also: *constraint force.*

inverse dynamics The problem of computing the forces and torques that would have to act on a collection of bodies, in order to yield given motion. See also: *forward dynamics.*

inverse kinematics The problem of computing the series of positioning transformations to apply to a collect of bodies, in order to yield a given configuration. See also: *forward kinematics, kinematic loop.*

kinematic Relating to motion without regard to considerations of inertia or force. See also: *dynamic.*

kinematic loop An arrangements of objects such that there is a circular dependency in the positions of the objects; for example, object A is connected to object B, which is connected to object C, which is connected to object A. The inverse kinematics problem (q.v.) posed by a kinematic loop in general yields a system of simultaneous non-linear equations, which is not easily solved.

least-squares The "least-squares solution" to a linear equation $\mathbf{A}\vec{x} + \vec{b} = 0$ is not an actual solution to the equation, but a value that is "closest" to a solution, in that it minimizes the deviation from 0. That is, it is the solution to the problem: given matrix $\mathbf{A}$ and vector $\vec{b}$, find the vector $\vec{x}$ that minimizes $|\mathbf{A}\vec{x} + \vec{b}|^2$. See also: *singular-value decomposition.*

linear system of equations A linear system of equations is a collection of linear equations involving the same variables, which can be expressed as a multidimensional equation $\mathbf{A}\vec{x} + \vec{b} = 0$. Typically, we are given a matrix $\mathbf{A}$ and a vector $\vec{b}$, and we want to find a vector $\vec{x}$ such that the equation is satisfied. If matrix $\mathbf{A}$ is non-singular, the analytic solution is $\vec{x} = \mathbf{A}^{-1}\vec{b}$. If the matrix is singular, the system may be overdetermined (q.v.) or underdetermined (q.v.). See also: *linear-system solver, least-squares.*

linear-system solver A subroutine that numerically solves a linear system of equations. $\mathbf{A}\vec{x} + \vec{b} = 0$, given a matrix $\mathbf{A}$ and a vector $\vec{b}$. Linear-system solvers typically arrive at a solution without explicitly inverting $\mathbf{A}$, for reasons of speed or numerical stability, or to handle equations with singular matrices. See also: *singular-value decomposition.*

linked system A linked system is a type of computer graphics model, in which rigid objects (typically rods) are connected via joints with various degrees of freedom. [Armstrong and Green 85] and

[Isaacs and Cohen 87] discuss modeling techniques for linked systems.

mass As in Newtonian mechanics, symbolized by m. See Appendix A.1.

momentum (linear) As in Newtonian mechanics, symbolized by $\vec{p}$. See Appendix A.1.

measure of the state A function that takes as input the state of a model; many different input states may yield the same value of the measure. The constraint deviation functions (q.v.) are measures of the state. See also: *state of the model.*

model A model is a representation of something. Typically, the model is a mathematical or computer analog of some (perhaps hypothetical) real-world object or objects. For our work, the model consists of the physical state of all the bodies, plus all the forces and torques that act on them. See also: *modeling.*

modeling The process of creating a model. See also: *physically-based modeling, traditional modeling, rendering.*

nail See *control point.*

net force The various forces on a rigid body can be combined linearly into a single "net force" that yields the same effect on the center of mass of the body. (The forces acting on a body also contribute to a "net torque" (q.v.)). See Appendix A.1. See also: *Euler's principle of superposition.*

net torque Each force on a body yields a torque on the body. We can also allow "pure torques" to act on the body. The various torques on the body can combined linearly into a single "net torque" that yields the same rotation of the body. See Appendix A.1. See also: *Euler's principle of superposition.*

Newton's laws of motion The basic laws of classical dynamics, that are encapsulated in the expression $F = ma$. See also: *Newtonian mechanics, classical dynamics.*

Newtonian mechanics See *classical dynamics..*

non-constraint force (torque) Any force (torque) in the model other than the constraint forces; the net force on a body is broken down into the constraint forces and the net non-constraint force (Sec. 3.3.2). See also: *external applied force (torque).*

orientation matrix A rotation matrix that describes the orientation of a body; the matrix transforms vectors from body coordinates to world coordinates. See Appendix A.1.

overdetermined system An overdetermined system of linear equations is one for which there is no solution. This arises because the individual equations are contradictory. If a linear system of equations is overdetermined, the matrix is singular. See also: *linear system of equations, least-squares.*

penalty method An optimization technique in which an error term is to be minimized by adding a "force" term that is proportional to the amount of error. This technique does not work well if there are other forces or momenta affecting the error. See Sec. 2.2.1.

physically-based modeling A modeling technique in which various physical parameters are incorporated into the model, allowing the behavior of the model to be simulated according to known natural laws.

primitive body In the modeling system (Ch. 5), a rigid body. The primitive bodies are the building blocks with which other objects are constructed.

quaternion A four-component mathematical quantity, that can be used to represent rotation. See Appendix B.1.

radius vector The vector from the center of mass of a body to a point in the body. See Fig. A.2.

render See *rendering.*

rendering Rendering is the process of creating an image of a computer graphics model.

"retaining" force The force which must be exerted on a body in order to keep it in place, if there is motion or other forces that are pulling on the body. See also: *constraint force.*

rigid body A body that moves as a unit. The body can not be deformed or decomposed. The dynamic state of a rigid body is completely described by its mass, rotational inertia, position, orientation, linear momentum, and angular momentum (see Fig. A.1).

rotational inertia tensor As in Newtonian mechanics, symbolized by $\mathbf{I}$; a tensor that describes how the angular momentum of a body

relates to its angular velocity. See Appendix A.1.

self-assembly When the modeler creates a compound object, the components of the model need not be created in their correct positions; the object will "assemble itself" as the constraint forces act on the components, pulling them into place.

simulation The process of using numerical techniques to solve equations embodying natural law, so as to yield behavior of a computer model that is analogous to the behavior that the real-world objects being modeled would exhibit. The simulation procedure for classical dynamics is described in Appendix A.

singular matrix A singular matrix is a matrix that can not be inverted: It is non-square, or there are linear dependencies among the rows or columns of the matrix. See *linear system of equations.*

singular-value decomposition (SVD) For any matrix **A**, the singular-value decomposition of **A** is the collection of three matrices **U**, **V**, and **D**, such that the columns of **U** are mutually orthogonal, the columns of **V** are mutually orthogonal, **D** is diagonal, and $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$. When computing the SVD of a matrix numerically, elements of **D** that are too small relative to the largest element of **D** are zeroed. The pseudoinverse of **A** can be computed via $\mathbf{A}^+ = \mathbf{V}\mathbf{D}^{-1}\mathbf{U}^T$, where we only use the terms corresponding to non-zero elements of D. Given a linear system $\mathbf{A}\vec{x} + \vec{b} = 0$, the value $\vec{x} = \mathbf{A}^+\vec{b}$ is: the solution to the system, if **A** is non-singular; the least-squares solution, if the system is overdetermined; or the smallest solution, if the system is underdetermined. See [Press et. al. 86].

sparse A matrix **A** is sparse if most of its elements are zero. Various numerical techniques exist that can handle sparse matrices quickly. See [Press et. al. 86]..

spatial state See *state of the model.*

square A linear system of equations is square if it has the same number of equations as unknowns; i.e. the matrix is square.

state of the model By "state of the model," we generally refer to the collection of the dynamic states of the rigid bodies. More completely, the state of the model includes the forces and torques acting in the model. The *spatial state* refers to the collection of positions and orientations of the bodies. See also: *measure of the state.*

state variable One of the quantities that describes the state of a rigid body: mass, rotational inertia, position, orientation, momentum, and angular momentum.

SVD See *singular-value decomposition.*

time constant The *time constant* (τ) for a constraint is a parameter that controls the rate of assembly; See Eqn. 3.4.

traditional modeling By "traditional modeling," we refer to the common computer-graphics modeling technique, which models the spatial state of bodies, and animates them kinematically. See also: *modeling.*

underdetermined system An underdetermined linear system of equations is one for which there is more than one solution; if such is the case, there is always an infinite space of solutions. A system may be underdetermined because some of the equations are redundant, or because some of the variables are redundant. If a linear system of equations is underdetermined, the matrix will be singular. See also: *linear system of equations, singular value decomposition.*

velocity By "velocity of a body", we are referring to the velocity of the center-of-mass of the body (see Appendix A.1). The velocity of a control-point on a body is computed based on the velocity of the body, the angular velocity of the body, and the radius vector to the control point (see Appendix B.3).

world coordinates A fixed global coordinate system which is used as the reference frame for classical dynamics. Also referred to as "lab coordinates." See also: *body coordinates.*

Chapter 1

Introduction

1.1 Computer Graphics Modeling

Computer graphics "picture making" can be divided into two parts: *modeling* and *rendering* (see Fig. 1.1). To create an image, we first create a *model*, a mathematical representation of the object or objects being imaged. The model must include all aspects of the objects that we wish to have affect the final image. Once we have created the model, we *render* its image; typically, by computing the interaction of light with the elements of the model, and projecting the results onto a two-dimensional "film."

The computer graphics modeling technique most commonly used today, which we shall refer to as the "traditional" technique, is purely spatial — the models represent the shapes and positions of objects. The traditional model consists of a collection of objects, with a description of the position and orientation of each object, as well as parameters describing the shape of each object (see Fig. 1.2). For animations, the models are parametrized by time.

This traditional approach to modeling is powerful and easy to use. Unfortunately, as models become complex, traditional modeling becomes overly cumbersome—specifying each detail by hand is tedious at best, and if complex relationships between objects are to be maintained as objects animate, specifying the details by hand becomes virtually impossible. Furthermore, if we wish traditional models to exhibit physically realistic behavior, it is up to the human modeler/animator to create this behavior by hand. The traditional modeling & animation tools ignore physical realism, tending to impart "puppet-like" behavior to models.

This thesis presents a modeling technique, called "Dynamic Constraints," that alleviates some of the problems of the traditional modeling technique. "Dynamic Constraints" is based on four principles:

- *Generality:* A model is built from a collection of primitive physically-based elements.
- *Geometric Constraints:* A model is constructed by applying constraints to the objects, starting from an initial configuration of the primitive elements. A model is also positioned and animated through constraints.
- *Newtonian Mechanics:* Each primitive element is a rigid body whose motion is due to the effects of inertia and forces and torques acting on the body. Many of the forces and torques are externally applied; other forces and torques, however, are derived from the geometric constraints.
- *Equivalence of Modeling and Animation:* The temporal behavior of physically based objects is bound

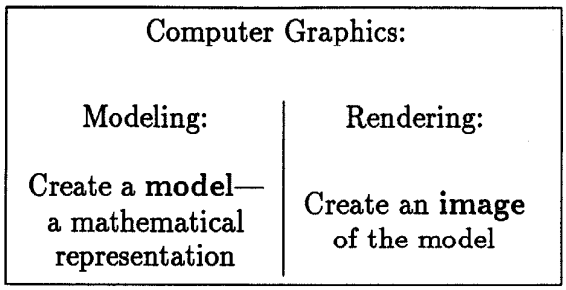


Figure 1.1: Breakdown of Computer Graphics "picture making"

This thesis is accompanied by a videotape that includes several sequences demonstrating "Dynamic Constraints" [Caltech '87 Demo Reel].

1.2 Physically-Based Modeling

By "Physically-Based" modeling, we refer to techniques that include into models not only spatial in-

"Traditional" Modeling:

```
SPHERE AT [3.2 12.9 17.32]
CUBE AT [2.5 -6.2 1.11]
    ROTATE [45.2°, 2.1°, 3.2°]
⋮
```

"Traditional" Animation:

```
Frame 1:
    SPHERE AT [3.2, 12.9, 17.32]
    CUBE AT [2.5 -6.2 1.11]
        ROTATE [45.2°, 2.1°, 3.2°]
    ⋮

Frame 2:
    SPHERE AT [3.5, 11.9, 17.35]
    CUBE AT [2.5 -6.3 1.31]
        ROTATE [44.2°, 2.6°, 4.0°]
    ⋮

⋮
```

Figure 1.2: "Traditional" Modeling: The "traditional" computer-graphics model is in essence a list of shapes, with the configuration of each shape. For animation, the list of shapes and configurations is varied as a function of time or "frame number".

formation, but also various physical parameters. The behavior of the models is then simulated according to well-known natural physical laws. Thus, as opposed to "traditional" modeling, in which the behavior of the model is created manually by the modeler, in physically-based modeling, realistic behavior is inherent in the model.

This thesis focuses on models created from primitive rigid elements; the models should behave in accordance with classical rigid-body dynamics (see Fig. 1.3). We thus incorporate into our models physical characteristics such as mass and momentum, and simulate the behavior according to Newton's Laws of Motion (see Fig. 1.4).

1.2.1 Physically-Based Modeling vs. Traditional Modeling

If we were to replace our traditional modeling techniques with simulation of rigid-body motion, we would be "buying" realistic behavior at the expense of ease-of-use. The traditional model is easy to use

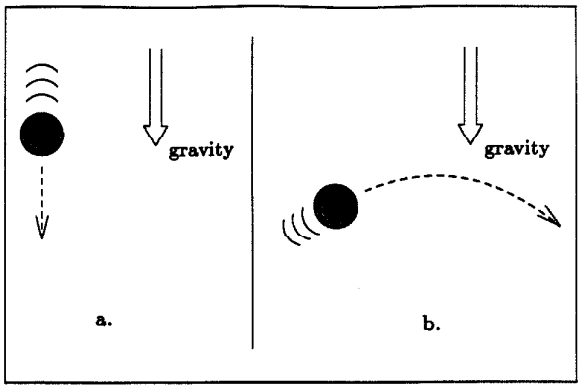


Figure 1.3: Physically-based Modeling: A physically-based modeling technique includes physical parameters into the model, and simulates the model's behavior according to natural law. In this thesis, all primitive bodies are rigid elements with parameters such as mass an momentum, and they move in accordance with Newton's laws. e.g.: (a) A ball released in gravity falls; (b) A ball thrown in gravity moves in an arc.

because of its "put that there" methodology: the modeler has direct control over the positions and motions in the model. For a physically-based simulation, the modeler must adjust parameters such as forces and torques in order to control positions and motions in the model—not an easy task.

To make physically-based modeling be a viable technique, we must make it easy to use: We must provide a mechanism that gives the modeler the ability to explicitly control the positions and motions in the model. Such a mechanism would automatically adjust the forces in the model to yield the behavior requested by the modeler. "Dynamic Constraints" provides such a mechanism.

1.2.2 Dynamic Constraints vs. Simulation

It is worth emphasizing the contrast between simulation and Dynamic Constraints (see Fig. 1.5). When we do simulation, we start with a model, and determine its behavior according to the well-known laws of motion. Dynamic constraints, however, addresses the inverse problem: starting with a partial description of the behavior we would like to see, we want to create a model whose behavior, when simulated, matches the description.

Physical Parameters for Rigid-Body Dynamics:	m — Mass I — Rotational Inertia Tensor $\vec{x}$ — Position $\mathbf{R}$ — Orientation $\vec{v}$ — Linear Velocity $\vec{p}$ — Linear Momentum $\vec{\omega}$ — Angular Velocity $\vec{L}$ — Angular Momentum $\vec{F}$ — Force $\vec{T}$ — Torque	Equations of Rigid-Body Dynamics: $\vec{v} = \frac{1}{m}p$ $\vec{\omega} = \mathbf{I}^{-1}\vec{L}$ $\frac{d}{dt}\vec{x} = \vec{v}$ $\frac{d}{dt}\mathbf{R} = \omega^*\mathbf{R}$ $\frac{d}{dt}\vec{p} = \vec{F}$ $\frac{d}{dt}\vec{L} = \vec{T}$

Figure 1.4: Physical parameters and equations of classical rigid-body dynamics. See Appendix A for a detailed discussion.

Dynamic Constraints
vs.
Simulation

Simulation:
Given a model,
Compute its behavior

Dynamic Constraints:
Given a description of desired behavior,
Create a model,
Use simulation to compute the behavior.

Figure 1.5: Dynamic Constraints addresses a problem that is the inverse of simulation—that of model creation. We are given a description of desired aspects of the model's behavior; we must create a model that will, upon simulation, behave in accordance with the desires.

1.3 Constraint-Based Control of Models

A geometric constraint is a restriction on the "legal" arrangements of positions and orientations of objects in a model. For example, the modeler may require that a certain point on an object stay fixed in space (Fig. 1.6), or that two objects remain in contact with each other (Fig. 1.7). A constraint may explicitly depend on time; for example, Fig. 1.8 illustrates a constraint requiring a point on an object to follow a predefined path. Fig. 1.9 illustrates a constraint involving the orientation of an object. In general, the "Dynamic Constraints" technique supports constraints that can be expressed as equations of the form

$$D(\vec{x}_1, \mathbf{R}_1, ..., \vec{x}_N, \mathbf{R}_N, t) = 0$$

where D is some twice-differentiable function, the $\vec{x}_i$'s and $\mathbf{R}_i$'s are positions and orientations of the bodies in the model (see Fig. 1.4), and t is time.

1.3.1 Constraint-Based Modeling vs. Traditional Modeling

Notice that constraint-based modeling provides a superset of the functionality of "traditional" modeling. We can directly control the exact positions and orientations of the model elements; the "traditional" modeling command "place object A at position X with orientation R" becomes "constrain object A to be at position X with orientation R." However, with constraint-based control, we are not required to specify this level of detail.

Thus constraint-based modeling allows the modeler to focus attention on "high-level" concerns, such as relationships between objects in the model, descending into detailed "low-level" positions and orientations of individual objects only when it is needed.

1.3.2 Self-Assembly of Models

Notice that, as illustrated in Fig. 1.7 and Fig. 1.6, compound objects can be built from component elements; the configuration of the components is described via constraints. The components of the model can be initially disparate; the constraints cause *self-assembly* of the resulting compound object.

1.4 Related Work

Much of the material in this thesis has been published in abbreviated form as [Barzel and Barr 88].

[Witkin, Fleischer, and Barr 87] uses "energy" constraints to assemble 3D models, for changing the shape of parametrically-defined primitive objects.

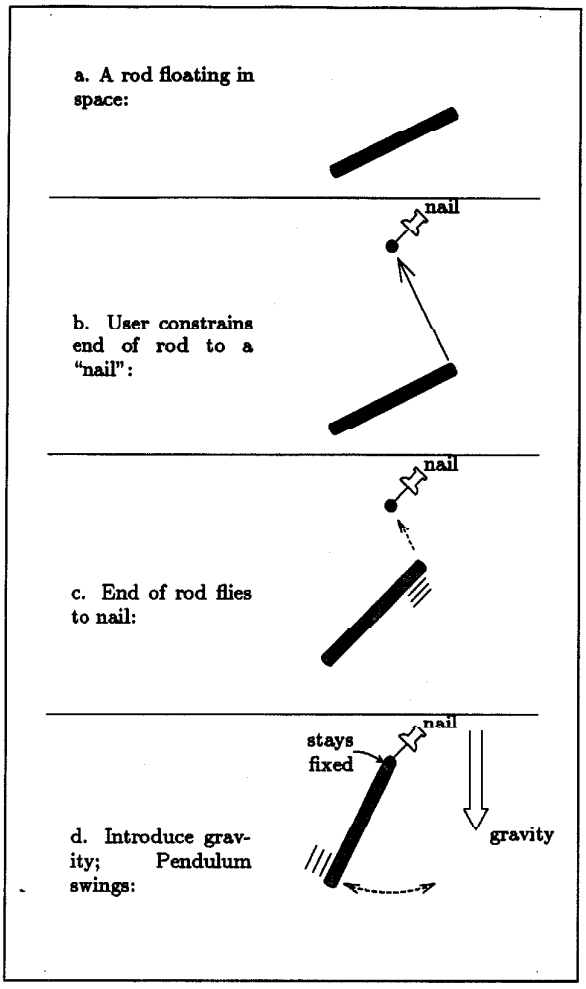


Figure 1.6: A "point-to-nail" constraint. A user creates a pendulum by fixing an endpoint of a rod at some location in space. The constraint causes the rod to "fly" into place, assembling the pendulum.

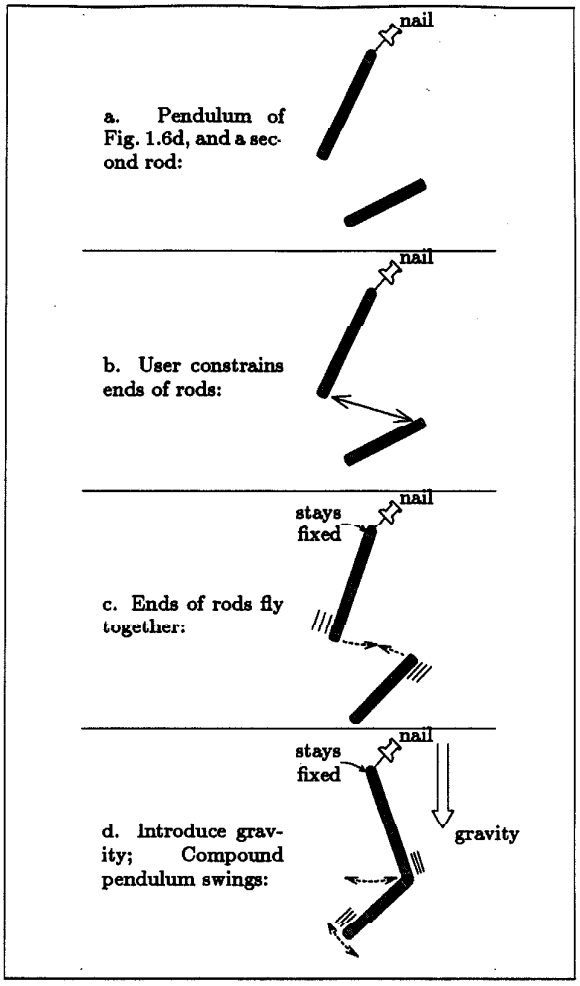


Figure 1.7: A "point-to-point" constraint. A user adds a second rod to the pendulum of Fig. 1.6, to create a compound pendulum. Appendix F.2 shows frames from an animation that demonstrates a compound pendulum being built in this way.

This work is not concerned with dynamic mechanical simulation of models. [Platt and Barr 88] uses augmented lagrangian constraints in the physical simulation of flexible objects. [Isaacs and Cohen 87] does physical simulation of rigid bodies, for the special case of linked systems without closed kinematic loops. They share our emphasis on ease of modeling, and also use an inverse-dynamics formulation to control the models' behavior. [Wilhelms and Barsky 85] utilizes physically based modeling, but has a reduced emphasis on control.

The inverse dynamics approach is new to computer graphics; however, inverse dynamics problems are not uncommon in physics and engineering. A work dealing with analytic solutions to inverse dynamics problems is [Galiullin 84].

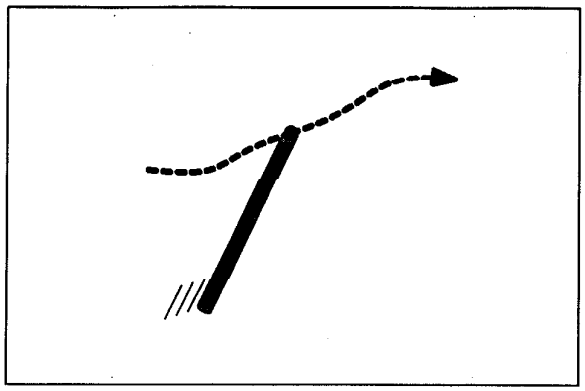


Figure 1.8: A "point-to-path" constraint. This constraint pulls objects along user-specified paths.

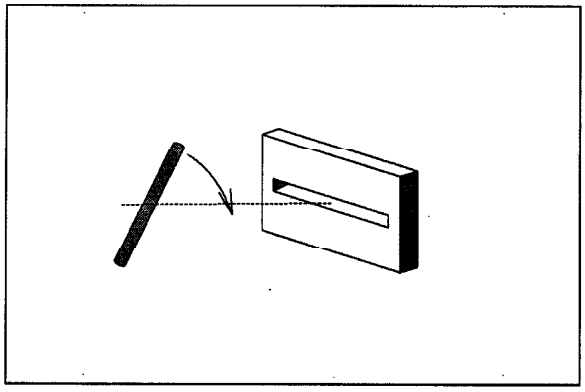


Figure 1.9: An "orientation" constraint. We rotate the rod to make its axis parallel to the slot axis.

Chapter 2

Dynamic Constraints: A Technique for Meeting Constraints while Simulating Rigid-Body Dynamics

2.1 "Constraint Forces"

Rigid-body dynamics permits control over a model's behavior via the forces and torques in the model. Accordingly, we implement constraints by introducing forces into the model. These "constraint forces" act on the constrained bodies, and are calculated to meet the constraints: If the constraints aren't met, the constraint forces will pull the bodies into place; if the constraints are met, the constraint forces will ensure that the constraints are maintained.

If a compound object is specified via constraints, there is a physical interpretation to the corresponding constraint forces: The constraint forces represent the *internal forces* that hold together the "real-world" object that has been modeled. For example, Fig. 2.1 illustrates the constraint forces in effect in the pendulum of Fig. 1.7; these are the same forces that are be applied by the hinges in a "real-world" pendulum.

The constraint forces that assemble objects don't correspond to pre-existing forces in the "real world"; they are merely our means for getting the elements of the object where we want them to be. However, the constraint forces represent real-world forces which could be used to assemble real-world objects. For example, the figures in Appendix F.7 show frames from an animation demonstrating the self-assembly of space structures [Barr, Von Herzen, Barzel, and Snyder 87].

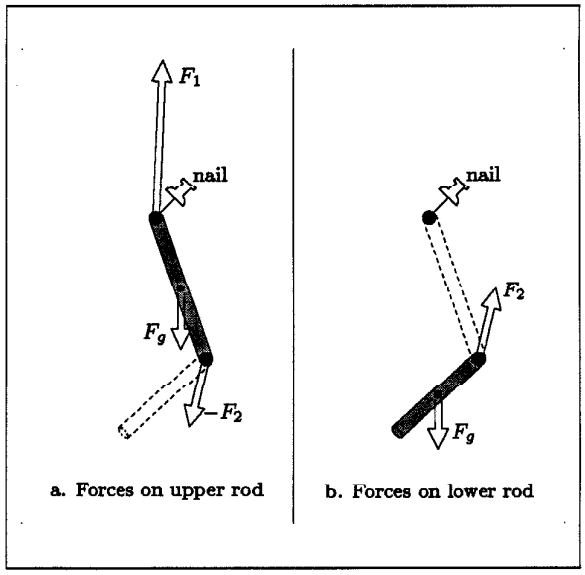


Figure 2.1: Constraint forces holding together the compound pendulum of Fig. 1.7d. The constraint forces represent the internal forces of an idealized pendulum. (a) shows forces on the upper rod, (b) shows forces on the lower rod: F_g is gravity pulling down on rods. F_1 is the "point-to-nail" constraint force on the upper rod, holding it at the nail. F_2 is the "point-to-point" constraint force on the lower rod, holding it to the upper rod; $-F_2$ is the reaction force on the upper rod.

2.2 Computing Constraint Forces

The key to successfully using a constraint-force technique is the determination of the constraint forces. We will discuss an intuitive approach ("rubber bands") that is unsatisfactory, in order to motivate the subtler *inverse dynamics* technique.

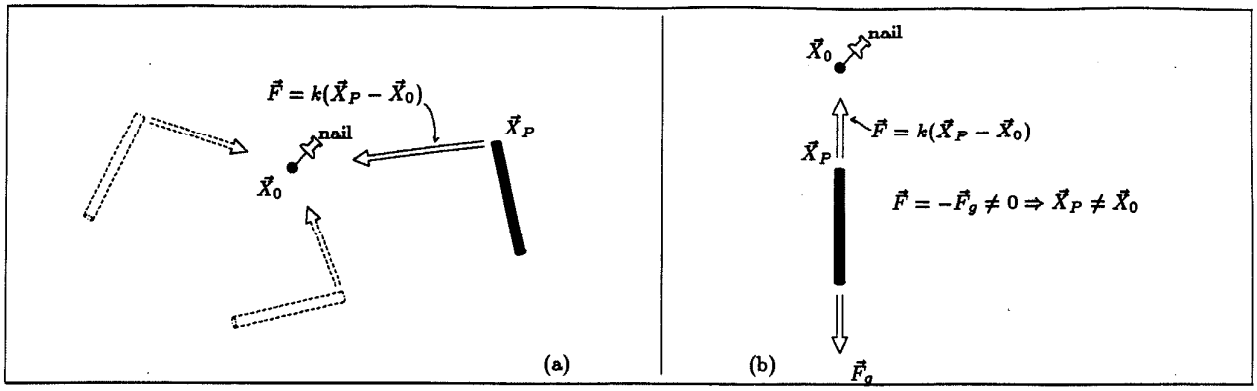


Figure 2.2: Attempting to use a "rubber band" to meet a "point-to-nail" constraint. (a) Rubber band force is proportional to the difference between the point and nail positions. (b) Rubber band cannot provide a "retaining" force and keep constraint met. For example, to suspend the rod from the nail in the presence of gravity, the rubber band must provide an upward force $\vec{F}$ equal and opposite to the gravitational force $\vec{F}_g$; the rubber band will only provide such a force when the point is away from the nail. See animation frames in Appendix F.1.

2.2.1 Motivational History: Rubber Bands (Penalty Method)

At first glance, it might seem that rubber bands could be used to provide the constraint forces. For example, we could try to meet a "point-to-nail" constraint (Fig. 1.6) by metaphorically stretching a rubber band between the nail and the body: When the point is far from the nail, we pull the point strongly toward the nail, and when the point is close to the nail, we pull less strongly (see Fig. 2.2a). This is known as the "penalty method" for optimization.

Unfortunately, in experimenting with the penalty method, several ways in which it falls short become apparent):

- Damping must be added to keep the system from oscillating.
- Different body masses and inertias will require different force and damping constants.
- The rubber bands will not maintain the constraint if other forces act on the bodies, or if the bodies are moving (see Fig. 2.2b). To maintain the constraint in such situations requires the application of a "retaining" force—but the rubber band applies no force when the constraint is met.
- If, to try to keep the rubber bands from pulling apart, we make them stronger, the resulting system of differential equations becomes stiffer (thus harder to simulate), and sufficient external force will pull them apart anyway.

Appendix F.1 shows frames from a sequence illustrating the results if rubber bands are used to implement constraint forces, to build the pendulum of Fig. 1.7. The animated sequence can be seen on the videotape [Caltech '87 Demo Reel].

2.2.2 Constraint-Force Requirements

From our experience with rubber bands, we have developed several requirements for the constraint force calculation. It should:

- Work for arbitrary rigid bodies, in arbitrary states. The calculation should take into account the state of each body, including its mass and rotational inertia as well as its linear and angular momentum, so that no case-by-case "tuning" is needed.
- Meet constraints smoothly. We want to meet the constraints without artifacts such as oscillation.
- Never "pull apart." We must be able to apply a "retaining" force to hold the constraint met. The force calculation should take into account other forces acting on the bodies.

Appendix F.2 shows frames from a sequence in which "Dynamic Constraints" are used to build the pendulum of Fig. 1.7. This sequence demonstrates that the "Dynamic Constraints" do in fact meet the above requirements. The animated sequence can be seen on the videotape [Caltech '87 Demo Reel].

2.2.3 Inverse Dynamics

If we are given the forces which act on a collection of objects, we can easily solve the *forward-dynamics* problem—that of determining the objects' behavior—through simulation. This is what we did when we attempted to meet constraints with rubber bands.

However, the constraint-force problem is an *inverse-dynamics* problem: We are given aspects of the behavior that we want the bodies to exhibit, and we must determine forces that will yield such behavior. A classical inverse-dynamics problem in physics is Newton's problem, which led to his derivation of the law of gravitational attraction: Given the observed motion of the planets, determine the force that causes such motion [Galiullin 84]. Our constraint-force problem is more general, in that we would like to meet arbitrary constraints; unlike Newton, however, we are not interested in an analytic solution, but rather in a numerical technique for computing the forces. [1]

We break down the constraint-force problem into two parts: manipulating objects to meet a constraint that is initially unmet, and adjusting the constraint forces so that the constraint is maintained.

Constraint Force to Meet a Constraint

Fig. 2.3 shows a constraint force being used to meet a constraint. Notice that, in addition to pulling the object into place, the constraint force must reverse itself as the constraint nears completion, to prevent overshoot and oscillation.

Notice that the problem of meeting a constraint is actually very loosely specified: How quickly should the constraint be met? Along what path should the object move? For our solution, as we shall see in Ch. 3, the "deviation" of the constraint decays exponentially, with a user-specified time constant.

Constraint Force to Maintain a Constraint

Fig. 2.4 shows a constraint force maintaining a constraint as an object moves under the influence of other forces. Notice that the constraint force must continually adapt to the state of the model as a whole, in order to keep the constraint from pulling apart.

The problem of determining the force to maintain a constraint often has a unique solution, in which the constraint forces provide the internal forces that hold together an object.

[1] Once we have determined the forces, we then solve the forward-dynamics problem to yield the complete behavior of the model.

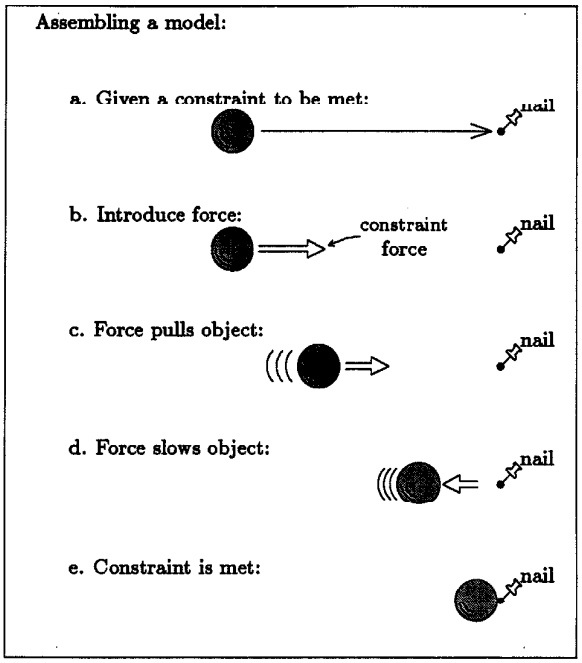


Figure 2.3: Meeting a Constraint With Forces. The constraint force pulls the ball towards the nail, then brings the ball to rest at the nail.

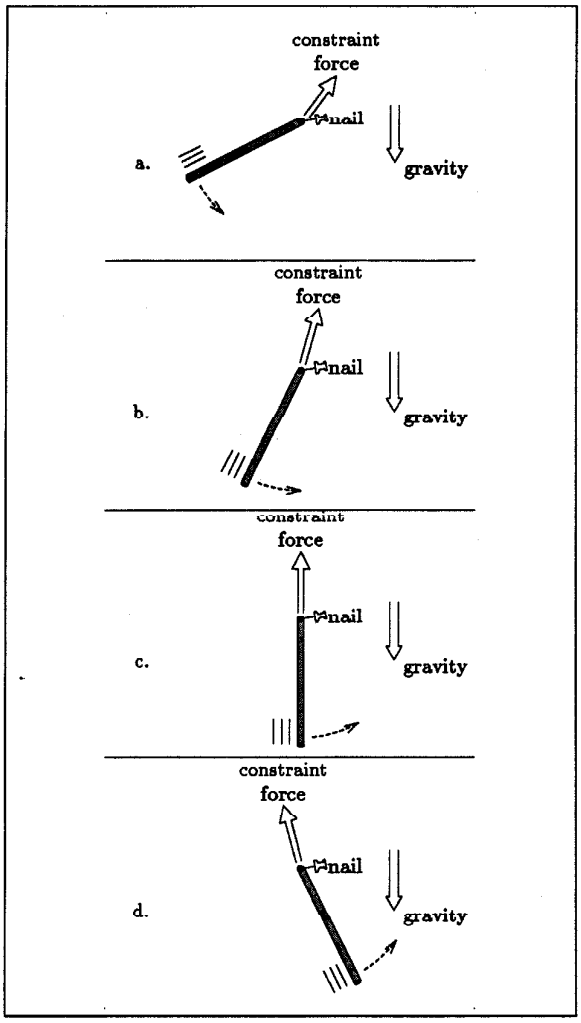


Figure 2.4: Maintaining a Constraint. (a-d) Constraint force adapts to hold constraint even as object moves and other forces act on it. The constraint force pulls up, to counteract gravity, and sideways, to keep the pendulum's inertia from flinging it sideways off the nail.

Chapter 3

Constructing a Constraint-Force Equation

Notes for this chapter:

- Appendix A lays out our notation and formulation of rigid-body mechanics.
- All terms are defined in the List of Mathematical Symbols (p. xi).
- Fig. 3.6 summarizes the constraint-related terms that we will define.
- All expressions are in world coordinates, unless otherwise specified.

This chapter presents the derivation of the "constraint-force equation," a linear equation whose solution yields the constraint forces that meet the collection of constraints in a model.

The inverse dynamics problem for constraint forces is summarized in Fig. 3.1: Given a constraint on a body or bodies, we are to determine forces that will pull the objects to meet the constraint, and will hold the constraint met.

The solution method for the inverse dynamics problem is broken down into four steps:

1. *Creating a Measure of the State of the Model.* We encapsulate those aspects of the model's state that are of interest to us, by creating a function $\vec{D}(\vec{x}_1, \mathbf{R}_1, \ldots)$ that measures the "deviation" of a constraint: $\vec{D}(\vec{x}_1, \mathbf{R}_1, \ldots) = 0$ when the constraint is met.
2. *Specifying Desired Behavior of the Measure.* We want $\vec{D}(\vec{x}_1, \mathbf{R}_1, \ldots)$ to decay from its initial value down to 0.
3. *Determine Influence of Forces on Behavior.* This is done through differentiation and application of the laws of motion.
4. *Find Forces that Yield Desired Behavior.* We combine the results of steps 1, 2 & 3 to derive a "constraint-force equation" that can be solved to yield the constraint forces.

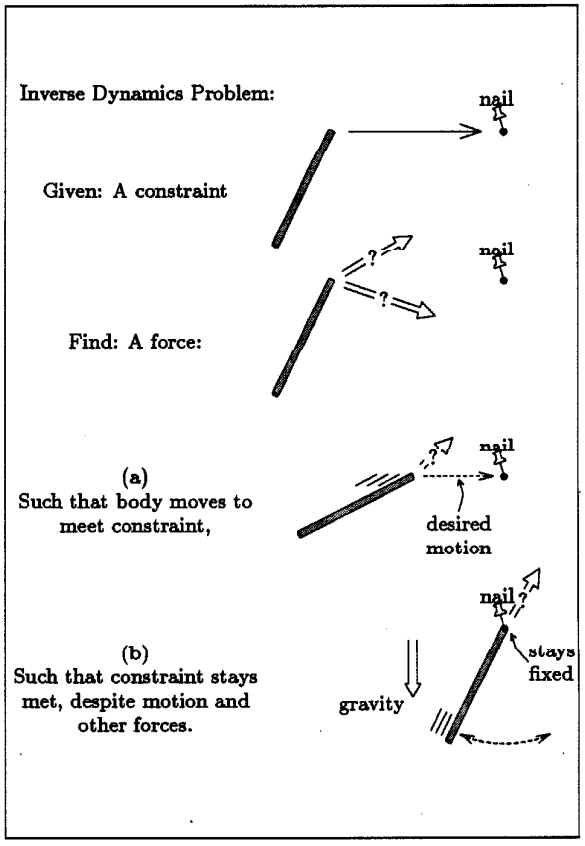


Figure 3.1: The inverse dynamics problem for dynamic constraints.

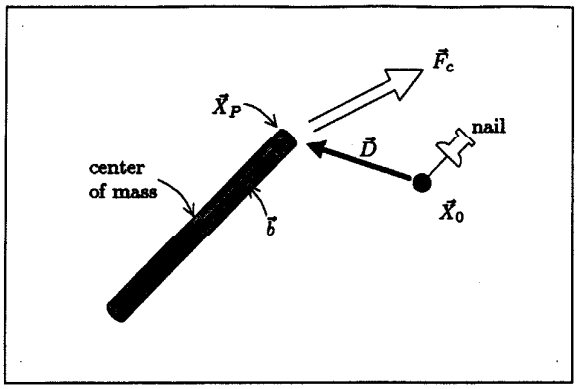


Figure 3.2: "Point-to-nail" constraint. The "deviation" measure is $\vec{D}(\mathcal{Y}) = \vec{X}_P(\mathcal{Y}) - \vec{X}_0$: when the point is at the nail, we have $\vec{D}(\mathcal{Y}) = 0$. The constraint force is $\vec{F}_c$, yielding a torque of $\vec{b} \times \vec{F}_c$.

3.1 *Step 1:* Defining the Constraint "Deviation" Function D

For a given type of constraint, we define a function "$\vec{D}$" which measures the "deviation" of the model from a state in which the constraint is met. That is, with

$$\begin{aligned} \vec{x}_i &= \textit{position of ith body (Appendix A.1)}, \\ \mathbf{R}_i &= \textit{orientation of ith body (Appendix A.1)}, \\ \mathcal{Y}_s(t) &= \left\{ \begin{array}{l} \textit{spatial state} \\ \textit{of the model} \end{array} \right\} \\ &= \{\vec{x}_1, \mathbf{R}_1, \ldots, \vec{x}_N, \mathbf{R}_N\} \end{aligned}$$

we define:

$$\vec{D}(\mathcal{Y}_s, t) = 0 \Longleftrightarrow \textit{constraint is met.} \qquad (3.1)$$

$\vec{D}(\mathcal{Y}_s, t)$ may be a vector, with any number of components; the constraint is met when all components are 0. The "point-to-nail" example below defines a function $\vec{D}(\mathcal{Y}_s, t)$ that is a vector, while the "orientation" constraint in Appendix C.3.4 uses a function $\vec{D}(\mathcal{Y}_s, t)$ that is a scalar.

Note that many constraints have no explicit time dependency, in these cases, we write the deviation function as $\vec{D}(\mathcal{Y}_s)$.

Example: "Point-to-Nail" Constraint. For the "point-to-nail" constraint, we define $\vec{D}$ to be the vector of displacement of the constrained point "P" from the nail (see Fig. 3.2):

$$\begin{aligned} \vec{D}(\mathcal{Y}_s) &= \vec{X}_P(\mathcal{Y}_s) - \vec{X}_0 \\ &= \vec{x} + \mathbf{R}\vec{b}_{body} - \vec{X}_0 \end{aligned} \qquad (3.2)$$

A detailed discussion of this and several other constraints is given in Appendix C.3.

3.2 *Step 2:* Expressing a Desired Behavior of D over Time

3.2.1 Requirements

When the constraint is first introduced, at some time t_0, the constraint will typically not be met; we will have some non-zero value $\vec{D}_0 \neq 0$. We want to satisfy both parts of the inverse-dynamics problem (Fig. 3.1): meeting the constraint, and maintaining it. To meet the constraint, we would like $\vec{D}$ to decrease to 0 over some period of time. To maintain the constraint, we would like $\vec{D}$ to stay at 0 from then on.

In other words, we want the composite function $\vec{D}(\mathcal{Y}_s(t), t)$ to be a continuous function such that

$$\begin{aligned} \vec{D}(\mathcal{Y}_s(t_0), t_0) &= \vec{D}_0 \\ \vec{D}(\mathcal{Y}_s(t), t) &= 0, t > t_1, \text{ for some } t_1 > t_0.^1 \end{aligned} \qquad (3.3)$$

Note that, as discussed in Sec. 2.2.3, other than the above requirements, we are free to choose any behavior that we find convenient.

[1] Strictly speaking, we don't need $\vec{D} = 0$ exactly; we only need $\vec{D} = 0$ within the error tolerances of our numerical techniques.

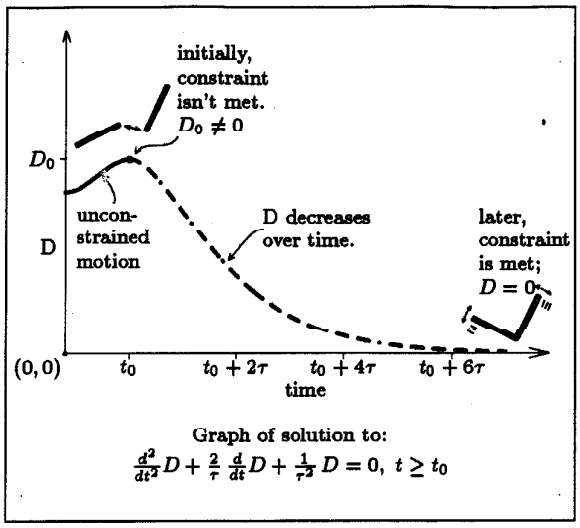


Figure 3.3: D evolving over time. We have selected a second-order differential equation to describe D as a function of time. The solution yields the behavior required by the inverse dynamics problem—the constraint "deviation" decays down to 0, assembling the model, then remains at 0, maintaining the constraint. The rate of assembly is controlled by the time constant τ. See [Boyce and Deprima 77] for a discussion of second-order differential equations.

3.2.2 Desired Behavior Expressed as a Differential Equation

We now choose a specific behavior which meets the requirements of Eqn. 3.3: critically-damped decay, graphed in Fig. 3.3. This behavior is described by the equation:[2]

$$\begin{aligned} &\tfrac{d^2}{dt^2}\vec{D}(\mathcal{Y}_s(t),t) + \tfrac{2}{\tau}\tfrac{d}{dt}\vec{D}(\mathcal{Y}_s(t),t) + \tfrac{1}{\tau^2}\vec{D}(\mathcal{Y}_s(t),t) = 0, \quad t \geq t_0 \\ &\vec{D}(\mathcal{Y}_s(t_0),t_0) \quad = \vec{D}_0 \\ &\tfrac{d}{dt}\vec{D}(\mathcal{Y}_s(t_0),t_0) \quad = (\tfrac{d}{dt}\vec{D})_0 \end{aligned} \tag{3.4}$$

where τ is the *time-constant* of the motion. See [Boyce and Deprima 77] for a discussion of critical damping and second-order linear differential equations.

We have chosen critically damped decay because it:

- Meets the requirements of Eqn. 3.3.
- Allows C^1 continuity: Critical damping matches the both the initial value, $\vec{D}_0$, and the initial velocity, $(\frac{d}{dt}\vec{D})_0$, then adjusts the velocity to bring $\vec{D}$ to 0. This allows us to activate a constraint during the course of a simulation, without any discontinuities in the velocities of the bodies.
- Has an explicit control over the rate of assembly: The rate of assembly is determined by the constant τ.
- Brings $\vec{D}$ to 0 smoothly: $\vec{D}$ does not oscillate.
- Can be expressed conveniently: Eqn. 3.4 can be transformed into an equation in the unknown constraint force, as we shall see in steps 3 & 4.

Note that this choice of behavior requires that, if $\vec{D}()$ is multidimensional, all components of $\vec{D}()$ must decay simultaneously. In some cases this is overly restrictive, and can result in failure in meeting the constraint (see Sec. 4.2.2)..

3.3 *Step 3:* Determining Influence of Forces on Behavior of D

3.3.1 Differentiate to Find Dependence on Net Force

To describe the dynamic behavior of the "deviation" function $\vec{D}(\mathcal{Y}_s,t)$, we create two functions, $\vec{D}^{(1)}(\mathcal{Y},t)$ and $\vec{D}^{(2)}(\mathcal{Y},\vec{F},\vec{T},t)$, such that:

$$\left.\begin{aligned} \vec{D}^{(1)}(\mathcal{Y}(t),t) &\equiv \tfrac{d}{dt}\vec{D}(\mathcal{Y}_s(t),t), \\ \vec{D}^{(2)}(\mathcal{Y}(t),\mathcal{F}(t),\mathcal{T}(t),t) &\equiv \tfrac{d^2}{dt^2}\vec{D}(\mathcal{Y}_s(t),t), \end{aligned}\right\} t \geq t_0 \tag{3.5}$$

To construct these functions, we differentiate $\vec{D}(\mathcal{Y}_s(t),t)$ with respect time, using the identities $\frac{d}{dt}x = \frac{1}{m}\vec{p}$, etc. (Fig. A.1, Appendix A) to replace differentials of $\mathcal{Y}$ with functions of $\mathcal{Y}$, $\vec{F}$ or $\vec{T}$.

[2] Analytically, if $\vec{D}_0 \neq 0$ the solution to Eqn. 3.4 asymptotically approaches 0, but doesn't ever reach $\vec{D} = 0$. Numerically, however, we quickly reach $\vec{D} = 0$ within error tolerances.

$\vec{D}^{(2)}$ will depend linearly on $\mathcal{F}$ and $\mathcal{T}$, the net forces and torques on the constrained objects (see Appendix A). This is because $\vec{D}$ depends only on the positions and orientations ($\vec{x}$ and $\mathbf{R}$) of the bodies in the model; in differentiating $\vec{D}$, the second-order terms ($\vec{F}$ and $\vec{T}$) emerge linearly via the differentiation chain rule. We can thus express $\vec{D}^{(2)}$ in general as

$$\vec{D}^{(2)}(\mathcal{Y},\mathcal{F},\mathcal{T}) = \sum_{\text{bodies } i} (\mathbf{\Gamma}^i(\mathcal{Y})\vec{F}^i + \mathbf{\Lambda}^i(\mathcal{Y})\vec{T}^i) + \vec{\beta}(\mathcal{Y}) \tag{3.6}$$

where we have

d	— Number of components of $\vec{D}()$
$\vec{F}^i$	— Net force on body i
$\vec{T}^i$	— Net torque on body i
$\mathbf{\Gamma}^i(\mathcal{Y})$	— A $d \times 3$ matrix function of $\mathcal{Y}$
$\mathbf{\Lambda}^i(\mathcal{Y})$	— A $d \times 3$ matrix function of $\mathcal{Y}$
$\vec{\beta}(\mathcal{Y})$	— A d-component function of $\mathcal{Y}$

We have a different function $\mathbf{\Gamma}^i$ and $\mathbf{\Lambda}^i$ for each body i in the constraint.

Example: "Point-to-Nail" Constraint. For the "point-to-nail" constraint (Fig. 3.2, Eqn. 3.2), the quantities in Eqn. 3.5 and Eqn. 3.6 are:

$$\begin{aligned}
d &= 3 \\
\vec{D}(\mathcal{Y}) &= \vec{X}_P(\mathcal{Y}) - \vec{X}_0 \\
\vec{D}^{(1)}(\mathcal{Y}) &= \vec{v}_P(\mathcal{Y}) \\
\vec{D}^{(2)}(\mathcal{Y},\vec{F},\vec{T}) &= \vec{a}_P(\mathcal{Y},\vec{F},\vec{T}) \\
\mathbf{\Gamma} &= \tfrac{1}{m} \\
\mathbf{\Lambda} &= \mathbf{b}^{*T}\mathbf{I}^{-1} \\
\vec{\beta} &= (\mathbf{I}^{-1}(\vec{L}\times\vec{\omega}))\times\vec{b} \\
&\quad +\vec{\omega}\times(\vec{\omega}\times\vec{b})
\end{aligned} \tag{3.7}$$

A full derivation of these quantities is given in Appendix C.3.1. Note that although $\mathbf{\Gamma}$ is nominally a 3×3 matrix, in this example it is a scalar; it could of course be expressed in matrix form as a scaled identity matrix.

3.3.2 Separate Net Force into Constraint & Non-Constraint Forces

A body may have many forces and torques acting on it. These may have been introduced into the model due to constraints, or they may have been introduced explicitly by the modeler. Thus we classify the forces and torques in the model into "constraint" and "non-constraint." Gravity and friction are examples of non-constraint forces.

Grouping together the non-constraint forces and torques acting on a body, we define:

$$\begin{aligned}
\vec{F}_N(\mathcal{Y},t) &= \{\textit{Net non-constraint force on a body}\} \\
&= \vec{F}_{N_1} + \ldots + \vec{F}_{N_n} \\
\vec{T}_N(\mathcal{Y},t) &= \{\textit{Net non-constraint torque on a body}\} \\
&= \left\{\begin{array}{c}\textit{net}\\ \textit{non-constraint}\\ \textit{pure torques}\end{array}\right\} + \left\{\begin{array}{c}\textit{torques due}\\ \textit{to non-constraint}\\ \textit{forces}\end{array}\right\} \\
&= \vec{T}_{N_1} + \ldots + \vec{T}_{N_m} + \vec{b}_1 \times \vec{F}_{N_1} + \ldots + \vec{b}_n \times \vec{F}_{N_n}
\end{aligned} \tag{3.8}$$

where we have

$\vec{F}_{N_1},\ldots,\vec{F}_{N_n}$	— Non-constraint forces
$\vec{b}_1,\ldots,\vec{b}_n$	— Radii at which forces are applied
$\vec{T}_{N_1},\ldots,\vec{T}_{N_m}$	— Non-constraint pure torques

Each $\vec{F}_i, \vec{T}_i$, and $\vec{b}_i$ is some modeler-specified function of $\mathcal{Y}$ and t.

A single constraint might result in forces and torques on several bodies. For example, the "point-to-point" constraint in Appendix C.3.3 applies an equal-and-opposite force pair to the constrained bodies, and each body experiences a torque, if its constraint force is not applied at its center of mass.

In general, for each constraint we define a vector quantity $\vec{F}_c$ with f components; the force and torque on each constrained body are derived by linear operations on $\vec{F}_c$.[3] We thus define:

$\vec{F}_{c_j}$	—	"constraint force" for constraint j; An f_j-dimensional vector
f_j	—	number of dimensions of $\vec{F}_{c_j}$
$\mathbf{G}^i_j$	—	A $3 \times f_j$ matrix for body i.
$\mathbf{H}^i_j$	—	A $3 \times f_j$ matrix for body i.

$$\begin{aligned}
\left\{\begin{array}{l}\textit{force on body } i\\ \textit{due to constraint } j\end{array}\right\} &= \mathbf{G}^i_j \vec{F}_{c_j} \\
\left\{\begin{array}{l}\textit{torque on body } i\\ \textit{due to constraint } j\end{array}\right\} &= \mathbf{H}^i_j \vec{F}_{c_j}
\end{aligned} \tag{3.9}$$

Notice that for a constraint that is met via torques only (e.g. the "orientation" constraint in Appendix C.3.4), $\mathbf{G}$ will be 0. Notice also that we can define $\mathbf{G}$ and $\mathbf{H}$ for every body/constraint pair—if a body is not involved in a constraint, $\mathbf{G}^i_j$ and $\mathbf{H}^i_j$ will be 0.

We combine the constraint and non-constraint

[3] Although $\vec{F}_c$ is not necessarily, strictly speaking, a force (but rather a quantity from which we compute the constraint force and torque), we colloquially refer to $\vec{F}_c$ as the "constraint force" for its constraint.

terms, to express the net force and torque on a body:

$$\begin{aligned}\vec{F}^i &= \textit{Net force on body } i \\ &= \left\{\begin{matrix}\textit{constraint}\\ \textit{forces on } i\end{matrix}\right\} + \left\{\begin{matrix}\textit{net}\\ \textit{non-constraint}\\ \textit{forces on } i\end{matrix}\right\} \\ &= \Big(\sum_{\text{constraints } j} \mathbf{G}^i_j \vec{F}_{c_j}\Big) + \vec{F}^i_N \\ \vec{T} &= \textit{Net torque on body } i \\ &= \left\{\begin{matrix}\textit{constraint}\\ \textit{torques}\end{matrix}\right\} + \left\{\begin{matrix}\textit{net}\\ \textit{non-constraint}\\ \textit{torques}\end{matrix}\right\} \\ &= \Big(\sum_{\text{constraints } j} \mathbf{H}^i_j \vec{F}_{c_j}\Big) + \vec{T}^i_N\end{aligned} \tag{3.10}$$

Example: "Point-to-Nail" Constraint. For the "point-to-nail" constraint, we apply an arbitrary force to the body at the constrained point. $\vec{F}_c$ can be used directly as the constraint force; the accompanying torque would be equal to $\vec{b} \times \vec{F}_c$ (see Fig. 3.2). Thus the quantities in Eqn. 3.9 are:

$$\begin{aligned} f &= 3 \\ \mathbf{G} &= 1 \\ \mathbf{H} &= \mathbf{b}^* \end{aligned} \tag{3.11}$$

3.4 *Step 4:* Constructing The Constraint-Force Equation

3.4.1 Equation for a Single Constraint

We transform Eqn. 3.4 from a differential equation to an algebraic equation, by substituting in $\vec{D}^{(1)}()$ and $\vec{D}^{(2)}()$ of Eqn. 3.5, expanding $\vec{D}^{(2)}()$ as per Eqn. 3.6:

$$\sum_{\text{bodies } i} (\mathbf{\Gamma}^i \vec{F}^i + \mathbf{\Lambda}^i \vec{T}^i) + \vec{\beta} + \frac{2}{\tau}\vec{D}^{(1)} + \frac{1}{\tau^2}\vec{D} = 0, t \geq t_0 \tag{3.12}$$

We expand $\vec{F}^i$ and $\vec{T}^i$ as per Eqn. 3.10, yielding:

$$\begin{aligned}\sum_{\text{constraints } j}\sum_{\text{bodies } i} (\mathbf{\Gamma}^i \mathbf{G}^i_j + \mathbf{\Lambda}^i \mathbf{H}^i_j)\vec{F}_{c_j} \\ + \sum_{\text{bodies } i} (\mathbf{\Gamma}^i \vec{F}^i_N + \mathbf{\Lambda}^i \vec{T}^i_N) \qquad = 0, t \geq t_0 \\ + \vec{\beta} + \frac{2}{\tau}\vec{D}^{(1)} + \frac{1}{\tau^2}\vec{D}\end{aligned} \tag{3.13}$$

where we label terms for the ith body with superscript i's, and terms for the kth constraint with subscript k's. Notice that, to meet this one constraint, we must take into account the effect of all the constraint forces acting on the constrained bodies.

Example: "Point-to-Nail" Constraint. In Fig. 3.4, we illustrate a simple case: A single body, with a "point-to-nail" constraint acting at its center-of-mass, and with gravity. Since both the constraint force and gravity act on the ball's center-of-mass, there are no rotational terms. Thus we have:

$$\begin{aligned} \vec{b} = \mathbf{b}^* = \mathbf{\Lambda} = \vec{\beta} = \mathbf{H} = \vec{T}_N &= 0 \\ \vec{D} &= \vec{x} - \vec{X}_0 \\ \vec{D}^{(1)} &= \vec{v} \\ \vec{D}^{(2)} &= \tfrac{1}{m}\vec{F} \\ \mathbf{\Gamma} &= \tfrac{1}{m} \\ \mathbf{G} &= 1 \\ \vec{F}_N &= \vec{F}_g \end{aligned} \tag{3.14}$$

Substituting into Eqn. 3.13 gives:

$$\frac{1}{m}\vec{F}_c + \frac{1}{m}\vec{F}_g + \frac{2}{\tau}\vec{v} + \frac{1}{\tau^2}(\vec{x} - \vec{X}_0) = 0 \tag{3.15}$$

We easily solve for the constraint force:

$$\vec{F}_c = -\vec{F}_g - \frac{2}{\tau}m\vec{v} - \frac{1}{\tau^2}m(\vec{x} - \vec{X}_0) \tag{3.16}$$

Thus we see the constraint force has three components: One opposing the force of gravity, one opposing the ball's velocity, and one pulling the ball towards the nail. Fig. 3.4 illustrates the constraint force adapting to pull the ball to the nail, and bring it to rest. Once the ball is at rest at the nail, we have $\vec{x} - \vec{X}_0 = 0$ and $\vec{v} = 0$; so Eqn. 3.16 simplifies to $\vec{F}_c = -\vec{F}_g$, yielding a net force on the ball of $\vec{F} = \vec{F}_c + \vec{F}_g = 0$

3.4.2 Equation for Multiple Constraints

With several constraints in a model, each constraint yields a version of Eqn. 3.13. Thus, to meet all the constraints, we derive a set of simultaneous equations.

We duplicate Eqn. 3.13, for each constraint in the model:

$$\begin{aligned}\sum_{\text{constraints } j}\sum_{\text{bodies } i} (\mathbf{\Gamma}^i_k \mathbf{G}^i_j + \mathbf{\Lambda}^i_k \mathbf{H}^i_j)\vec{F}_{c_j} \\ + \sum_{\text{bodies } i} (\mathbf{\Gamma}^i_k \vec{F}^i_N + \mathbf{\Lambda}^i_k \vec{T}^i_N) \qquad = 0 \left\{\begin{matrix}\text{for all}\\ \text{con-}\\ \text{straints}\\ k\end{matrix}\right. \\ + \vec{\beta}_k + \frac{2}{\tau_k}\vec{D}^{(1)}_k + \frac{1}{\tau^2_k}\vec{D}_k\end{aligned} \tag{3.17}$$

where we label terms for the kth constraint with subscript k's. Writing this system of equations more compactly, as a multidimensional vector equation, we have the constraint force equation for the model:

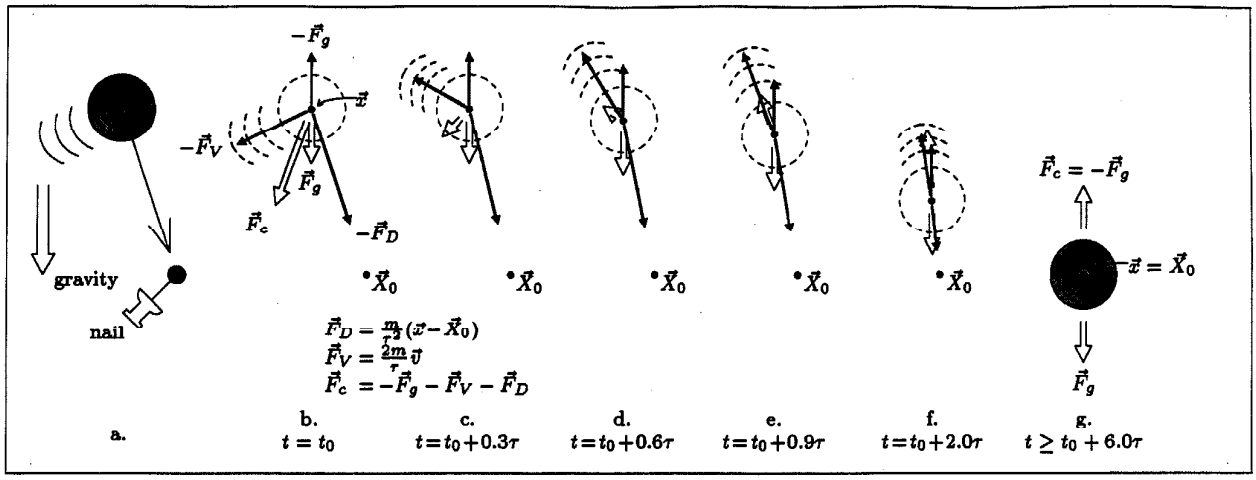


Figure 3.4: Constraint-force calculation for a "point-to-nail" constraint (details in Sec. 3.4): Constraint force has components opposing gravity ($-\vec{F}_g$), opposing motion ($-\vec{F}_v$), and pulling towards nail ($-\vec{F}_D$). (a) User specifies constraint at center of mass. (b) Constraint force initially pulls towards nail. (c-e) Once ball is moving towards nail, constraint force turns around. (f) Constraint force slows ball. (g) Steady-state: $\vec{F}_D = \vec{F}_V = 0$, $\vec{F}_c = -\vec{F}_g$. Net force on ball is 0; by Newton's first law, the ball remains at rest.

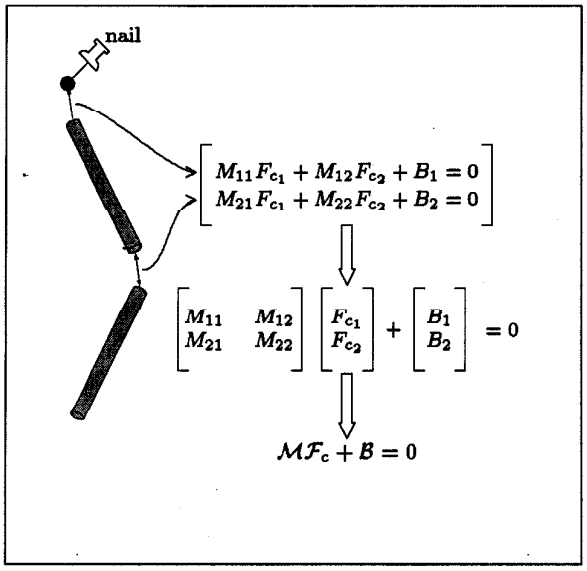


Figure 3.5: Multiple constraints: Each constraint contributes one line to the equation. The collection of constraints together yields a set of simultaneous linear equations, expressible as a linear matrix equation.

$$\boxed{\mathcal{M}\mathcal{F}_c + \mathcal{B} = 0} \tag{3.18}$$

where

$$\mathcal{M}_{kj} = \sum_{\text{bodies } i} (\mathbf{\Gamma}^i_k \mathbf{G}^i_j + \mathbf{\Lambda}^i_k \mathbf{H}^i_j)$$

$$\mathcal{F}_{cj} \quad -\vec{F}_{c_j}$$

$$\mathcal{B}_k = \sum_{\text{bodies } i} (\mathbf{\Gamma}^i_k \vec{F}^i_N + \mathbf{\Lambda}^i_k \vec{T}^i_N) + \vec{\beta}_k + \frac{2}{\tau_k}\vec{D}^{(1)}_k + \frac{1}{\tau_k^2}\vec{D}_k$$

Fig. 3.5 illustrates collecting individual constraint equations into the multidimensional vector equation. Notice that the $[k,j]$th element of $\mathcal{M}$ is a $d_k \times f_j$ matrix, and each element of $\mathcal{F}_c$ and of $\mathcal{B}$ is an f_j-component vector.

Fig. 3.6 summarizes the definitions of the terms that enter into Eqn. 3.18.

$\vec{D}_k$	A measure of "deviation" for constraint k: $$\vec{D}_k(\mathcal{Y}_s, t) = 0 \Longleftrightarrow \text{constraint } k \text{ is met}$$ $\vec{D}_k$ is a d_k-dimensional vector.
d_k	The number of dimensions of $\vec{D}_k$.
$\vec{D}_k^{(1)}$	The rate of change of $\vec{D}_k$: $$\vec{D}_k^{(1)}(\mathcal{Y}(t), t) = \frac{d}{dt}\vec{D}_k(\mathcal{Y}_s(t))$$ $\vec{D}_k^{(1)}$ is a d_k-dimensional vector.
$\vec{D}_k^{(2)}$	The acceleration of $\vec{D}_k$: $$\vec{D}_k^{(2)}(\mathcal{Y}(t), \mathcal{F}(t), \mathcal{T}(t), t) \equiv \frac{d^2}{dt^2}\vec{D}(\mathcal{Y}_s(t), t)$$ $\vec{D}_k^{(2)}$ will depend linearly on $\mathcal{F}$ and $\mathcal{T}$; thus: $$\vec{D}_k^{(2)} = \sum_{\text{bodies } i} (\mathbf{\Gamma}_k^i(\mathcal{Y})\vec{F}^i + \mathbf{\Lambda}_k^i(\mathcal{Y})\vec{T}^i) + \vec{\beta}_k(\mathcal{Y})$$ where we define: $\mathbf{\Gamma}_k^i$: A $d_k \times 3$ matrix corresponding to the net force on body i; we have one such for each body in constraint k. If body i is not involved in constraint k, $\mathbf{\Gamma}_k^i = 0$. $\mathbf{\Lambda}_k^i$: A $d_k \times 3$ matrix corresponding to the net torque on body i; we have one such for each body in constraint k. If body i is not involved in constraint k, $\mathbf{\Lambda}_k^i = 0$. $\vec{\beta}_k$: The part of $\vec{D}_k^{(2)}$ independent of $\mathcal{F}$ and $\mathcal{T}$; a d_k-dimensional vector.
$\vec{F}_N^i$	The net non-constraint force on body i
$\vec{T}_N^i$	The net non-constraint torque on body i
$\vec{F}_{c_j}$	The unknown "constraint force" of constraint j. An f_j-dimensional vector.
f_j	The number of degrees of freedom in the constraint force of constraint j.
$\mathbf{G}_j^i$	A $3 \times f_j$ matrix. The force on body i due to constraint j is given by $\mathbf{G}_j^i\vec{F}_{c_j}$.
$\mathbf{H}_j^i$	A $3 \times f_j$ matrix. The torque on body i due to constraint j is given by $\mathbf{H}_j^i\vec{F}_{c_j}$.
$\mathcal{F}_c$	The collection of constraint forces in the model; A multidimensional vector. The jth element of $\mathcal{F}_c$ is a f_j-component vector, given by $$\mathcal{F}_{cj} = \vec{F}_{c_j}$$
$\mathcal{M}$	A multidimensional matrix. The $[k,j]$th element of $\mathcal{M}$ is a $d_k \times f_j$ matrix, given by $$\mathcal{M}_{kj} = \sum_{\text{bodies } i} (\mathbf{\Gamma}_k^i\mathbf{G}_j^i + \mathbf{\Lambda}_k^i\mathbf{H}_j^i)$$ $\mathcal{M}_{kj}$ corresponds to the influence of the jth constraint-force on the kth constraint.
$\mathcal{B}$	A multidimensional vector. The kth element of $\mathcal{B}$ is a d_k-component vector, given by $$\mathcal{B}_k = \sum_{\text{bodies } i} (\mathbf{\Gamma}_k^i\vec{F}_N^i + \mathbf{\Lambda}_k^i\vec{T}_N^i) + \vec{\beta}_k + \frac{2}{\tau_k}\vec{D}_k^{(1)} + \frac{1}{\tau_k^2}\vec{D}_k$$ $\mathcal{B}_k$ corresponds to the deviation of the kth constraint, as well as the influence of motion and non-constraint forces on the constrained bodies.

Figure 3.6: Quantities associated with the constraint-force calculation. $\mathcal{Y}$, $\mathcal{F}$, and $\mathcal{T}$ are the state, net force, and net torque in the model (Eqn. A.5, Appendix A). $\mathcal{Y}_s$ is the spatial state of the model (Eqn. 3.1). Derivations of these quantities for various constraints are given in Appendix C.3.

Chapter 4

Solving the Constraint-Force Equation

To simulate a model with Dynamic Constraints, we do a standard numerical integration of the equations of classical rigid-body motion, as discussed in Appendix A. During the course of the simulation, we will have to compute the net force and torque on each body in the model, at arbitrary times, with the model in an arbitrary state. To compute the net forces and torques, we will have to set up and solve the constraint-force equation $\mathcal{M}\mathcal{F}_c + \mathcal{B} = 0$.

4.1 Setting Up The Equation

The procedure to set up and solve the constraint-force equation, and to compute the net forces in the model, is outlined in Fig. 4.1. Fig. 3.6 summarizes the various terms that were defined in Ch. 3.

We are given the functions $\vec{D}(\mathcal{Y}_s,t)$, $\vec{D}^{(1)}(\mathcal{Y},t)$, $\Gamma(\mathcal{Y},t)$, $\Lambda(\mathcal{Y},t)$, and $\vec{\beta}(\mathcal{Y},t)$ for each constraint in the model, as well as functions $\mathbf{G}(\mathcal{Y},t)$ and $\mathbf{H}(\mathcal{Y},t)$ for each constraint/body pair. Additionally, we know how to compute the external forces and torques in the model.

Thus, at any instant of time t, with the bodies in the model in state $\mathcal{Y}$, we can construct $\mathcal{M}$ and $\mathcal{B}$ as defined in Fig. 3.6. We then solve $\mathcal{M}\mathcal{F}_c + \mathcal{B} = 0$, to get the constraint forces. The constraint forces are combined with the external forces and torques to yield the net forces and torques on all the bodies.

4.2 Solving The Equation

Fig. 4.1 outlines the procedure to set up Eqn. 3.18, as well as solve it and compute the net force and torque on each body.

In step 3 of Fig. 4.1, we call a standard linear-system solver to solve Eqn. 3.18. There are many well-known methods for solving linear systems (see [Press et. al. 86], [Ralston and Rabinowitz 78]). We note some characteristics of $\mathcal{M}$ that should be taken into account when choosing a solution method:

- $\mathcal{M}$ is typically sparse. The $[k,j]$th entry in $\mathcal{M}$ is non-0 only if some body is influenced by both constraint k and constraint j.
- $\mathcal{M}$ is not necessarily square. A constraint may have $d \neq f$; for example, the "orientation" constraint (Appendix C.3.4) has $d = 1$ and $f = 3$, yielding a matrix which is "wider" than it is "tall."
- $\mathcal{M}$ may be singular, implying that Eqn. 3.18 is overdetermined or underdetermined.

We most often use singular-value decomposition (SVD) to solve Eqn. 3.18, because it robustly handles singularity and near-singularity, as well as non-square systems. However, SVD algorithms typically do not take advantage of sparseness, and are typically relatively slow. The glossary (p. xv) gives a brief description of SVD; see [Press et. al. 86] for a more complete introduction to SVD. [1]

One should also note that in the course of a simulation, we will have to solve the constraint-force equation repeatedly. Typically, each instance will be "near" the preceding ones; we are thus looking into ways to calculate the SVD quickly, by using a previous decomposition as an approximate initial guess for the new SVD.

[1] The SVD code given in [Press et. al. 86], which is based on code in the references found there, has a bug in its splitting of the bidiagonal matrix, causing incorrect decomposition in some cases. A matrix that elicits this bug is:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

We have re-implemented the algorithm in order to fix this bug. Additionally, the NAG Fortran Library [NAG] includes an implementation of SVD that does not suffer from this bug.

4.2.1 Underdetermined Equations

The constraint-force equation (Eqn. 3.18) will sometimes be underdetermined, thus having many solutions. This can occur, for example, when there are several constraints acting on a single body; it may be possible to vary some of the individual constraint forces without affecting the net torque or force on the object. An example is shown in Fig. 4.2a, in which the pair of forces labeled "V" yield the same net force ($= 2\vec{F}_V$) and torque ($= 0$) as the pair labeled "W".

There is no difficulty caused by having many solutions to Eqn. 3.18; we could use any solution, since they will all yield identical behavior. We might wish to use the solution which is smallest in magnitude, to avoid numerical difficulties; SVD yields this solution.

4.2.2 Overdetermined Equations

In Fig. 4.2b, the user has specified constraints which can not be met; there is no "correct" constraint force to be applied. In Fig. 4.2c, the specified constraints can be met, but not by moving the constrained point in a straight line; however, Eqn. 3.4 requires that the point move in straight line if the constrained point is initially at rest.[2]

For overdetermined systems, using the least-squares solution for the constraint forces typically yields "reasonable" behavior – the object typically assumes some intermediate configuration, for the case of Fig. 4.2b, or moves along the feasible path, for the case of Fig. 4.2c. SVD returns the least-squares solution for overdetermined systems.

4.2.3 Unbounded Solutions

It is possible for a deviation function $\vec{D}(\mathcal{Y}_s)$ to yield an equation $\mathcal{M}(\mathcal{Y})\mathcal{F}_c + \mathcal{B}(\mathcal{Y}) = 0$ that in turn results in a solution $\mathcal{F}_c(\mathcal{Y})$ with a singularity: near the singularity, $\mathcal{F}_c$ can be unboundedly large, and at the singularity, there is no solution. If the constraint tries to bring the model near the singularity, the force may grow so large that the numerical simulation will fail—even though the solution is valid analytically.

For example, consider the one-dimensional deviation function

$$\vec{D}(x) = x^2 \tag{4.1}$$

for which the constraint is met when $x = 0$. This function leads to the constraint-force equation:

$$\frac{2}{m}xF + 2v^2 + \frac{4}{\tau}xv + \frac{1}{\tau^2}x^2 = 0 \tag{4.2}$$

(where v is the one-dimensional velocity), resulting in

$$F = -\frac{mv^2}{x} - \frac{2mv}{\tau} - \frac{mx}{2\tau^2} \tag{4.3}$$

which blows up near $x = 0$ when $v \neq 0$. Since the constraint is trying to bring x to 0, the model will try to pass through states in which x is close to 0, with v non-zero, and the force will grow unboundedly, causing difficulty for the numerical O.D.E. solver.

To avoid unbounded solutions, one could either reformulate the deviation function (e.g. use $\vec{D}(x) = x$ in the above example), or shift the desired solution away from the pole (e.g. use $\vec{D}(x) = x^2 - \epsilon$ for some small ϵ). The latter approach has the drawback that it no longer exactly meets the constraints, analytically.

4.2.4 Discontinuous Solutions

It is possible to construct a deviation function $\vec{D}(\mathcal{Y}_s)$ such that the resulting constraint-force equation $\mathcal{M}(\mathcal{Y})\mathcal{F}_c + \mathcal{B}(\mathcal{Y}) = 0$ can be solved at all points in state space, but that the solution is discontinuous. This can cause difficulty for the numerical simulation.

For example, consider the one-dimensional deviation function

$$\vec{D}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \tag{4.4}$$

This constraint is met when $x \leq 0$; it represents an impenetrable wall at $x = 0$. As long as the body is to the left of $x = 0$, the constraint is met, and the least-squares solution gives a constraint-force of 0. If the body crosses over the $x = 0$ boundary, a non-0 constraint force is instantaneously introduced. The instantaneous change in force will likely cause the numerical simulation to fail, or at least to take extremely small steps.

Discontinuous forces arise in a physical model when rigid bodies collide. Appendix D discusses approaches to handling collisions within the dynamic-constraints framework.

[2] A partial solution to the problem of unrealizable paths is to use scalar constraint measures ($d = 1$). For example, the "point-to-nail" constraint could be redefined so that D is the distance from the point to the nail, rather than the vector separating the point and the nail.

```
PROCEDURE TO COMPUTE THE NET FORCE AND
TORQUE ON EACH BODY:

; 1. Compute net non-constr forces & torques
for each body i
    F⃗ⁱ_N = T⃗ⁱ_N = 0
    for each non-constr force j on body i
        compute force F⃗_Nj
        F⃗ⁱ_N += F⃗_Nj
        T⃗ⁱ_N += b⃗_j × F⃗_Nj
    end
    for each non-constr torque j on body i
        compute torque T⃗_Nj
        T⃗ⁱ_N += T⃗_Nj
    end
end

; 2. Set up constraint-force equation
initialize M to 0
for each constraint k
    compute β⃗_k, D⃗^(1)_k, and D⃗_k
    B[k] = β⃗_k + (2/τ_k) D⃗^(1)_k + (1/τ²_k) D⃗_k
    for each body i in constraint k
        compute Γⁱ_k and Λⁱ_k
        B[k] += Γⁱ_k F⃗ⁱ_N + Λⁱ_k T⃗ⁱ_N
        for each constraint j acting on i
            M[k,j] += Γⁱ_k Gⁱ_j + Λⁱ_k Hⁱ_j
        end
    end
end

; 3. Solve constraint-force equation (Eqn. 9.18)
F_c = solve(M,B) ;linear-system solver

; 4. Compute net forces and torques.
for each body i
    F⃗ⁱ = F⃗ⁱ_N
    T⃗ⁱ = T⃗ⁱ_N
    for each constraint j acting on i
        F⃗ⁱ += Gⁱ_j F_c[j]
        T⃗ⁱ += Hⁱ_j F_c[j]
    end
end
```

Figure 4.1: The procedure to compute the constraint forces and the net force and torque on each body. See discussion in Sec. 4.1. Note that the [k,j]th element of $\mathcal{M}$ is a $d_k \times f_j$ matrix, and the [k]th element of $\mathcal{B}$ is a d_k-dimensional vector. Rather than implementing $\mathcal{M}$ as a "nested" array, it can be flattened into a $(\sum_k d_k) \times (\sum_k f_k)$ array; similarly, $\mathcal{B}$ is formed by concatenating the individual vectors into one $(\sum_k d_k)$-dimensional vector.

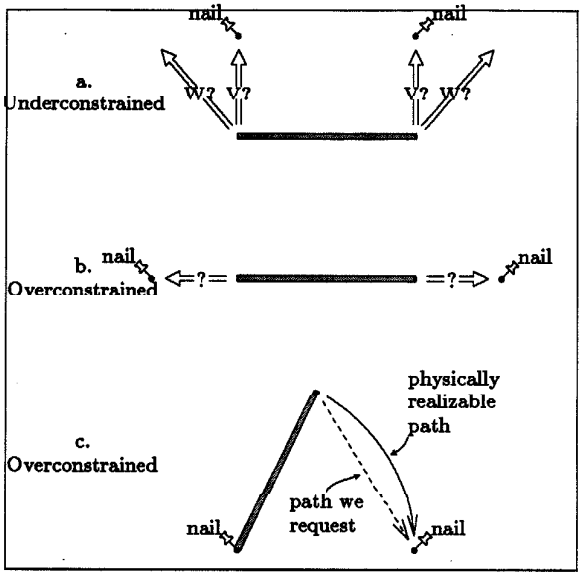


Figure 4.2: Under- and Overdetermined systems. (a) Underdetermined: Forces "V" and "W" yield the same net force. (b) Overdetermined: There is no way to meet both constraints. (c) Overdetermined: Both constraints could be met, but not via the path we have chosen.

Chapter 5

A Modeling System based on Dynamic Constraints

In this chapter, we provide a brief description of the "Dynamic Constraints" modeling system. This system was used to create the animations shown in Appendix F and [Caltech '87 Demo Reel].

5.1 Overview

Modeling with the "Dynamic Constraints" system consists of instantiating primitive bodies, connecting and controlling them with constraints, and influencing their behavior by explicitly applying forces. The modeling system thus has three libraries:

- *Primitive bodies:* A collection of rigid bodies, such as spheres, rods, tori, and more complex shapes, that are the component elements of models. The modeler specifies the body density, as well as specific parameters such as the length and radius of a rod. Each body type defines the quantities needed for physical simulation, such as the rotational inertia tensor for that body type.

- *External applied forces:* Forces (and torques) that the modeler can introduce into the model, including gravity, springs, and damping forces. Each force has parameters specific to the force, such as damping coefficients or spring constants. Using the terminology of Sec. 3.3.2, these are all "non-constraint" forces.

- *Constraints:* Various types of geometric constraints, such as the "point-to-nail" or "orientation" constraints described in Ch. 1 and derived in Appendix C.3.

It is fairly easy to add new entries to any of these libraries, in order to support new types of bodies, forces, or constraints.

To build a model, the modeler makes instances of objects, and runs the simulation. The modeler may also specify "timelines" of events to take place, such as creating or removing of instances, turning constraints or external forces on or off, or otherwise adjusting parameters.

When the modeler specifies a constraint, the system automatically generates the appropriate constraint forces and torques. These are added to the external applied forces and torques to yield the net forces and torques on the bodies.

5.2 Implementation

We describe here the high-level program structure; details of simulating Newtonian mechanics are given in Appendix A, and the procedure to calculate the constraint forces is given in Sec. 4.2.

Our modeling system is implemented in Common Lisp on Symbolics Lisp Machines, using Symbolics' object-oriented "Flavors" mechanism. The fundamental object classes we have defined are:

- *rigid-body:* a primitive body in the model. This class defines the functions and state variables needed for the dynamics calculation (see Appendix A), including a list of forces and torques acting on the body. There are subclasses for each type of body in our library; each subclass provides type-specific information, such as the rotational inertia tensor.

- *control-point:* a point on a body, or in space. A point on a body contains a reference to the body, as well as the position of the point in body-coordinates. A point in space defines its position, which can be constant, or a function of time. Forces and constraints are typically created by specifying the control-points at which

they act.

- *force:* a force being applied to a body. Each force contains a reference to a control point at which it is applied. There are subclasses for each type of force in our library; each subclass provides a function that computes the force.
- *constraint:* any type of dynamic constraint. There are subclasses for each type of constraint in our library. Each subclass provides the quantities needed to determine the constraint force (described in Fig. 3.6). Each constraint also keeps references to the bodies being constrained, and associates the appropriate forces with the bodies.

All objects handle a "draw" message, which displays the object in its current state. For debugging a model, we send the "draw" message to all objects, including forces, control points, and constraints; for producing an animation, we send the "draw" message only to bodies.

Some examples of subclasses are:

- *rod:* a subclass of *rigid-body*. This class provides the values specific to rods, e.g. the rotational inertia tensor (see Appendix A). The class also associates two control points with each rod, named "end1" and "end2", at the ends of the rod, and provides functions to access them.
- *nail:* a control point fixed at a location in space.
- *point-to-nail:* a subclass of *constraint*. Provides the functions which calculate the terms needed for a "point-to-nail" constraint (see Eqn. 3.2, Eqn. 3.7, Eqn. 3.11).

The addition of new types of bodies, forces, or constraints to the system merely requires the creation of an appropriate new subclass.

Currently, the user-interface is via the lisp environment; for example, the pendulum of Fig. 1.7 could be built via the series of commands:

```
; create bodies and control points
(make rod "upper-rod")
(make rod "lower-rod")
(make nail "nail" 0 0 100)
; specify constraints
(pt-to-pt (end1 "upper-rod") "nail")
(connect (end2 "upper-rod")
         (end1 "lower-rod"))
; add external forces
(gravity-on) ; apply gravity to each body
```

To animate a model once the instances are made, we simply iterate these steps:

- Simulate until end of frame (Appendix A).
- Send "draw" message to objects.

The implementation makes heavy use of a home-grown package of numerical routines, which include linear-system solvers, differential equation solvers, and the like; some useful references are [Press et. al. 86], [Golub and Van Loan 83], [Ralston and Rabinowitz 78], [Boyce and Deprima 77]. We also have embedded into lisp an extension to "Einstein Summation Notation" for mathematical expressions [Barr 83], [Misner, Thorne and Wheeler 73]; this makes it quite simple to create lisp functions by merely typing in the mathematical formulae using the same notation with which we derive them.

Chapter 6

Conclusion

6.1 Summary

We have developed a technique that facilitates the creation of computer models of dynamic systems composed ultimately of rigid bodies.

The technique, called "Dynamic Constraints" allows models to be built and manipulated through specification of geometric constraints on the rigid bodies' states. The constraints are met by introducing "constraint forces" into the model.

We have developed an inverse dynamics method for derivation of the "constraint-force equation" that is solved to yield the constraint forces: We start by defining a "deviation" function that measures whether a constraint is met; the modeler's request for a constraint is expressed as a desired exponential decay of the "deviation" function. The behavior is described by a second-order differential equation, which we then transform into the linear constraint-force equation.

6.2 Advantages

Several advantages, or otherwise good features, of the dynamic constraints technique are:

- The technique provides for the self-assembly of objects, and allows constraints to be changed and turned on and off over the course of a simulation.
- The technique allows the newtonian simulation to be done in Euclidean coordinates, rather than in state space (as in, e.g., [Armstrong and Green 85], [Isaacs and Cohen 87]).
- The technique has no difficulty with closed kinematic loops (as do, e.g., [Isaacs and Cohen 87], [Gerard and Maciejewski 85]).
- It is easy to add new constraints, of a variety of types.
- The technique is reasonably graceful for over-constrained models.

6.3 Disadvantages

Some of the drawbacks of the technique are:

- It is possible to define constraints that can't be solved, as discussed in (Sec. 4.2.3).
- The technique tries to meet constraints by moving along straight lines, which can cause unnecessary overdeterminism in the equations (Sec. 4.2.2).
- A set of geometric constraints does not always completely specify an object; alternate "isomeric" solutions can exist. See discussion in Appendix E.
- The technique is numerically intensive.

6.4 Future Work

Further work we are interested in pursuing includes:

- *Expanding the constraint library.*
- *Object Intersection.* Development of non-interpenetration constraints
- *Flexible bodies.* Incorporation of flexible-body simulation with dynamic constraint control [Platt and Barr 88], [Terzopoulos, Platt, Fleischer, and Barr 87].
- *Special-case models* Direct implementation of the equations of motion for common objects, such as the linked systems of [Isaacs and Cohen 87],

[Armstrong and Green 85]. Decreasing the number of constraints in the model speeds up the constraint-force calculation.

- *Constraints with inequalities.* A more general class of constraints is those that can be expressed as $\vec{D}(\mathcal{Y}_s, t) \geq 0$. This would allow constraints such as "make object A be higher than object B".

- *Constraints on velocity or acceleration.*

We are also looking forward to using the dynamic constraints modeling system as a tool in other research areas, such as molecular biology [Lengyel 87] and robotics.

Appendix A

Simulating Newtonian Mechanics

Fig. A.1 summarizes the equations of motion of a rigid body. A full discussion of rigid-body dynamics can be found in [Fox 67], [Goldstein 80].

A.1 Notes On The Equations Of Motion

We give some comments about the terms defined in Fig. A.1.

- **Mass m:** The mass m is a scalar quantity. m may in general be a function of the state of the system and/or time. For simplicity, in Appendix A.3 and Appendix B.3, we assume the mass of each primitive body is constant.

- **Rotational Inertia Tensor I:** $\mathbf{I}$ is a 3×3 matrix that determines the rotational behavior of a body, in that it defines the relationship between a body's angular velocity $\vec{\omega}$ and momentum $\vec{L}$. $\mathbf{I}$ is defined by: [1]

$$\mathbf{I} = \int_V (\mathbf{1}\vec{r}^2 - \vec{r}\vec{r})\rho(\vec{r})d\vec{r} \qquad \text{(A.1)}$$

  where $\vec{r}$ is the radius vector from the center of mass, and $\rho(\vec{r})$ is the density of the body. Note that for a rigid body with constant mass, $\mathbf{I}_{body}$ is constant. Note also that in Fig. A.1 we use the inverse $\mathbf{I}^{-1}$ rather than $\mathbf{I}$. Thus $\mathbf{I}^{-1}_{body}$ can be precomputed for each body; we convert to world coordinates using $\mathbf{R}$:

$$\mathbf{I}^{-1} = \mathbf{R}\mathbf{I}^{-1}_{body}\mathbf{R}^{\mathrm{T}} \qquad \text{(A.2)}$$

- **Position of Center of Mass $\vec{x}$:** $\vec{x}$ is a 3-space vector. For simplicity, we place the origin of the body coordinate system at the center of mass of the body (see Fig. A.2.)

[1] A discussion of the characteristics of $\mathbf{I}$ is beyond the scope of this paper; see [Fox 67], [Goldstein 80]. [Lien and Kajiya 84] gives an algorithm to compute $\mathbf{I}$ for arbitrary nonconvex polyhedra.

State variables of a body:

m	- Mass of the body
$\mathbf{I}$	- Rotational inertia tensor of the body
$\vec{x}$	- Position of the body center of mass
$\mathbf{R}$	- Orientation of the body
$\vec{p}$	- Momentum of the body
$\vec{L}$	- Angular Momentum of the body,

auxiliary variables:

$\vec{v} = \frac{1}{m}\vec{p}$ – Velocity of the body
$\vec{\omega} = \mathbf{I}^{-1}\vec{L}$ – Angular velocity of the body,

THE EQUATIONS OF MOTION:

$$\frac{d}{dt}\vec{x} = \vec{v}$$
$$\frac{d}{dt}\vec{p} = \vec{F}$$
$$\frac{d}{dt}\mathbf{R} = \omega^*\mathbf{R}$$
$$\frac{d}{dt}\vec{L} = \vec{T},$$

where:

$\vec{F}$ = net force on the body, *and*
$\vec{T}$ = net torque of the body.

Note: ω^* is the dual of $\vec{\omega}$ (see Appendix B.2).
Note: All expressions are in world coordinates.

Figure A.1: Summary of the equations of motion of a rigid body.

- **Orientation Matrix R:** $\mathbf{R}$ is a 3×3 matrix that transforms tensors from body-coordinates to world coordinates (see Fig. A.2). As we numerically integrate $\mathbf{R}$ (see Appendix A.3), numerical noise tends to cause $\mathbf{R}$ to drift away from a pure rotation, yielding noticeable skewing. This can be alleviated by using a feedback technique, as in [Barr 83]. Alternatively, we can represent the orientation as a quaternion $\mathbf{Q}$. The dynamic behavior of $\mathbf{Q}$ is given

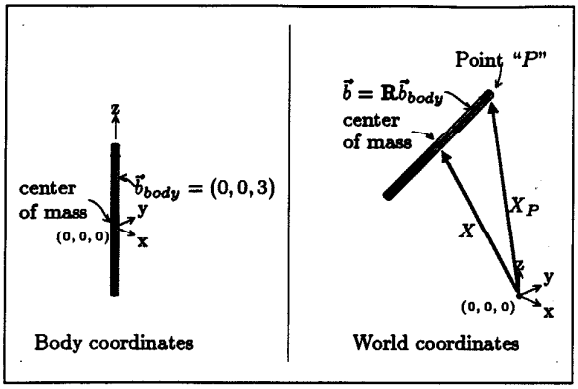


Figure A.2: A rigid body. Orientation matrix **R** transforms vectors from body coordinates to world coordinates.

by:[2]

$$\frac{d}{dt}\mathbf{Q} = \frac{1}{2}\vec{\omega}\mathbf{Q} \qquad \text{(A.3)}$$

(Quaternions and quaternion multiplication are summarized in Appendix B.1). We define **R** to be an auxiliary variable, which is computed from **Q**.

- **Linear Momentum $\vec{p}$:** $\vec{p}$ is a 3-space vector.
- **Angular Momentum $\vec{L}$:** $\vec{L}$, a 3-space vector, is the angular momentum of a body about its center of mass.
- **Linear Velocity $\vec{v}$:** $\vec{v}$, a 3-space vector, is the velocity of the center of mass of the body.
- **Angular Velocity $\vec{\omega}$:** $\vec{\omega}$, a 3-space vector, describes the direction and speed of rotation of the body: $\vec{\omega}$ lies along the instantaneous axis of rotation; the magnitude of $\vec{\omega}$ is the speed of rotation (revolutions per unit time) of the body.
- **Net Force $\vec{F}$ and Torque $\vec{T}$:** Euler's principle of superposition allows us to combine the forces applied to a body into an equivalent net force applied at the center of mass. Each force applied at a radius $\vec{b}$ from the center of mass contributes $\vec{b} \times \vec{F}$ to the net torque on the body; we can also have "pure torques" acting on the body. The net force and torque are thus:

$$\vec{F} = \sum_{\text{forces } i} \vec{F}_i$$

$$\vec{T} = \sum_{\text{forces } i} (\vec{b}_i \times \vec{F}_i) + \sum_{\text{torques } j} \vec{T}_j$$

[2] See [Misner, Thorne and Wheeler 73], page 1139.

A.2 Canonical O.D.E. Notation: $\frac{d}{dt}\mathcal{Y} = f(\mathcal{Y}, t)$

For brevity, we express the collection of equations in Fig. A.1 as an ordinary differential equation (O.D.E.), in canonical form. For a system consisting of single body A, we have:

$$\frac{d}{dt}Y^{\mathrm{A}} = f(Y^{\mathrm{A}}, \vec{F}^{\mathrm{A}}, \vec{T}^{\mathrm{A}}) \qquad \text{(A.4)}$$

where we define

$Y^{\mathrm{A}} = \{\vec{x}^{\mathrm{A}}, \mathbf{R}^{\mathrm{A}}, \vec{p}^{\mathrm{A}}, \vec{L}^{\mathrm{A}}\}$ —State of body A
$\vec{F}^{\mathrm{A}}$ —Net force on A
$\vec{T}^{\mathrm{A}}$ —Net torque on A

For a model consisting of a collection of bodies, we have:

$$\frac{d}{dt}\mathcal{Y} = f(\mathcal{Y}, \mathcal{F}, \mathcal{T}) \qquad \text{(A.5)}$$

where we define

$\mathcal{Y} = \{Y^{\mathrm{A}}, Y^{\mathrm{B}}, \ldots\}$ —State of the bodies
$\mathcal{F} = \{\vec{F}^{\mathrm{A}}, \vec{F}^{\mathrm{B}}, \ldots\}$ —Forces on the bodies
$\mathcal{T} = \{\vec{T}^{\mathrm{A}}, \vec{T}^{\mathrm{B}}, \ldots\}$ —Torques on the bodies

A.3 Implementation

A.3.1 Data Structures

The data structure for a rigid body should include:

- *Constant state variables:* m and $\mathbf{I}_{body}^{-1}$. These values are set when the body is created. Note the use of $\mathbf{I}_{body}^{-1}$, as discussed in Appendix A.1.
- *Differential state variables:* $\vec{x}$, **Q**, $\vec{p}$, and $\vec{L}$. These values are initialized when the body is created; thereafter they are computed by O.D.E integration, as described below. Note the use of **Q** as discussed in Appendix A.1.
- *Auxiliary variables:* $\vec{v}$, $\vec{\omega}$, and **R**. These are computed from the values of the other state variables.

The data structure for a force should include:

- *Force function:* Called to compute the force.
- *Body:* The body to which the force is applied.
- *Radius vector:* $\vec{b}_{body}$. The point on the body at which to apply the force; a constant vector in body coordinates.

A.3.2 Computing $\frac{d}{dt}\mathcal{Y}$

The O.D.E. solver (see below) requires us to provide a function $f(\mathcal{Y},t)$ that computes $\frac{d}{dt}\mathcal{Y}$. We are given an array $\mathcal{Y}$, a time value t, and an array $\mathcal{V}$ in which to store the result. The steps which we follow are:

1. Distribute the values from array $\mathcal{Y}$ into the bodies' state variables.[3]
2. Compute the bodies' auxiliary variables.
3. Compute the net force and torque on each object, as discussed in Sec. 4.2.
4. Compute the derivatives of each state variable of each body, using the equations of motion in Fig. A.1 and Eqn. A.3. The derivative values are concatenated into array $\mathcal{V}$.

A.3.3 Solving the O.D.E.

Numerical methods for solving first-order O.D.E.'s are well known; see Appendix A.3.4 for a brief discussion of the choice of solver.

An O.D.E. solver typically has the following features:

- Treats $\mathcal{Y}$ as a single long vector.
- Starts with an initial value of $\mathcal{Y}$ at time t_0
- Determines $\mathcal{Y}$ for $t > t_0$ by integrating $\frac{d}{dt}\mathcal{Y}$. The user supplies a function $f(\mathcal{Y},t)$ which computes $\frac{d}{dt}\mathcal{Y}$.
- Takes a "target" time t at which the value of $\mathcal{Y}$ is desired.

To create an animation, we must compute $\mathcal{Y}$ for each frame, at times t_0, $t_0+\Delta t$, $t_0+2\Delta t$, ... We thus perform the following steps.

1. Given the initial state of the bodies, at time $t = t_0$. Concatenate the values of differential state variables of all the bodies into a single 1-D array $\mathcal{Y}$.
2. Call the solver with "target" time $t+\Delta t$. The solver fills the array $\mathcal{Y}$ with its values at that time.
3. Distribute the values from array $\mathcal{Y}$ into the state variables of the bodies.
4. Draw the model.
5. Increment t by Δt, and iterate from step 2.

[3]Remember that $\mathcal{Y}$ as passed to $f(\mathcal{Y},t)$ is just a "hypothetical" state—the solver typically explores the state space, calling this function for various values of $\mathcal{Y}$ and t, which may not lie on the actual solution to the O.D.E.

A.3.4 Choosing an O.D.E. Solver

In our experience, we have generally been able to make do with adaptive step-size forward methods, such as an Adams-Bashforth predictor corrector (see [Ralston and Rabinowitz 78]). Using such methods, however, means that we have had to be careful not to create models whose differential equations were too stiff; otherwise the ODE solver would take unacceptably small steps, or even become unstable.

For stiff models, we would have to use implicit techniques, such as Gear's method ([Gear 71]). Unfortunately, implicit techniques generally require the creation of the jacobian matrix of $\frac{d}{dt}\mathcal{Y}$ — this typically involves many evaluations of $\frac{d}{dt}\mathcal{Y}$, each of which can be very expensive. For a stiff system, implicit methods can take much larger steps than forward methods, so despite the added cost per step, the implicit methods might be more practical.

One thing to keep in mind is that, for animation, we want to sample the solution $\mathcal{Y}(t)$ at regular points in time—at the end of each frame. Most differential equation solvers work by creating local (typically polynomial) approximations to the solution. Such solvers may use arbitrary step sizes and step past frame boundaries; we can interpolate back to get the state of the model at the frame boundaries.[4] In our experience, during the "easy" parts of a simulation, the solver will step over several frames at a time. Some solvers, such as Bulirsch-Stoer (see [Press et. al. 86]) can reputedly take huge steps, to compute the value of $\mathcal{Y}(t)$ for large t, but they don't provide information about the value of $\mathcal{Y}(t)$ in the intervening times. Without the intervening values of $\mathcal{Y}(t)$, the solution is less useful for animation.

Another consideration is that at discontinuities or singularities in $f(\mathcal{Y},t)$, a unique solution to $\frac{d}{dt}\mathcal{Y} = f(\mathcal{Y},t)$ is not guaranteed. It is not surprising that numerical differential equation solvers have great difficulty bridging such discontinuities. This issue arises in Sec. 4.2.4 and Appendix D.

[4]Note that the state of the model at each frame includes the velocities of the objects, allowing for rendering with motion-blur.

Appendix B

Mathematical Details

B.1 Quaternions

We present here a brief overview of quaternions and relevant formulae, using the notation of [Shoemake 85]. Quaternions are used to represent orientations of bodies (Appendix A.1).

B.1.1 Definition

A quaternion is a four-component object, having a scalar part and a vector part. Thus, with

$$\begin{aligned} s &= a\ scalar, \\ \vec{v} &= a\ vector, \end{aligned}$$

we write

$$q = [s, \vec{v}] \tag{B.1}$$

B.1.2 Quaternion-Quaternion Multiplication

To multiply quaternion q_1 by quaternion q_2, the formula is

$$\begin{aligned} q_1 &= [s_1, \vec{v}_1] \\ q_2 &= [s_2, \vec{v}_2] \end{aligned}$$

$$q_1 q_2 = [s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2, (s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2)] \tag{B.2}$$

B.1.3 Quaternion-Vector Multiplication

To multiply a vector by a quaternion (as in Eqn. A.3), we consider the vector to be a quaternion with zero scalar part:

$$\begin{aligned} \vec{b} &= a\ vector \\ q &= [s, \vec{v}],\ a\ quaternion \end{aligned} \tag{B.3}$$

$$\begin{aligned} \vec{b}\, q &= [0, \vec{b}][s, \vec{v}] \\ &= [-\vec{b} \cdot \vec{v}, s\vec{b} + \vec{b} \times \vec{v}] \end{aligned}$$

B.1.4 Quaternion Inverse

The inverse of a quaternion has its vector-part negated, and is divided by the square of the magnitude:

$$\begin{aligned} q &= [s, \vec{v}] \\ |q|^2 &= s^2 + |\vec{v}|^2 = 1 \\ q^{-1} &= \tfrac{1}{|q|^2}[s, -\vec{v}] \\ q\, q^{-1} &= q^{-1} q = [1, \vec{0}] = 1 \end{aligned} \tag{B.4}$$

B.1.5 Rotation by a Quaternion

A quaternion q can represent a rotation by θ around an axis $\vec{r}$ (a unit vector):

$$q = [cos(\frac{\theta}{2}), sin(\frac{\theta}{2})\vec{r}] \tag{B.5}$$

We rotate a vector $\vec{b}$ by:

$$Rotate(\vec{b}) = q\, \vec{b}\, q^{-1} \tag{B.6}$$

Notice that the magnitude of the quaternion is irrelevant to the rotation; rotations can be represented as points on the unit 4-sphere. Notice also that diametrically opposed points on the 4-sphere represent the same rotation.

B.1.6 Conversion to Rotation Matrix

To convert a normalized quaternion q to the corresponding rotation matrix M, the formula is[1]

$$q = [w, (x, y, z)], where\ w^2 + x^2 + y^2 + z^2 = 1$$

$$\mathrm{M} = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2wz & 2xz + 2wy \\ 2xy + 2wz & 1 - 2x^2 - 2z^2 & 2yz - 2wx \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix} \tag{B.7}$$

[1] The matrix given here is the transpose of that in [Shoemake 85]. This matrix is suitable for multiplying on the left, with column vectors; that is, we have $q\vec{b}q^{-1} = \mathrm{M}\vec{b}$

B.2 Dual of a vector

The *dual* of a vector $\vec{b}$ is the antisymmetric matrix $\mathbf{b}^*$:

$$\text{With } \vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}, \text{ define } \mathbf{b}^* = \begin{bmatrix} 0 & b_3 & -b_2 \\ -b_3 & 0 & b_1 \\ b_2 & -b_1 & 0 \end{bmatrix}$$

For any vectors $\vec{b}$ and $\vec{a}$, we have the following identities:

$$\begin{aligned} \mathbf{b}^*\vec{a} &\equiv \vec{b} \times \vec{a} \\ \mathbf{b}^{*\mathrm{T}}\vec{a} &\equiv \vec{a} \times \vec{b} \\ \mathbf{b}^{*\mathrm{T}} &\equiv -\mathbf{b}^* \\ \mathbf{b}^*\vec{b} &\equiv 0 \end{aligned}$$

B.3 Behavior of a Point

Consider a point "P" which is fixed relative to a rigid body (Fig. A.2). We define $\vec{b}_{body}$ to be the vector from the center-of-mass of the body to P, expressed in the body's home coordinates; since the point is fixed, $\vec{b}_{body}$ is constant. We would like to derive expressions for the position, velocity, and acceleration of P. We assume that the mass of the body is constant.

We will need to know the derivative of $\mathbf{I}^{-1}$. Remember that since the body is rigid, $\mathbf{I}_{body}^{-1}$ is constant:

$$\begin{aligned} \mathbf{I}^{-1} &= \mathbf{R}\mathbf{I}_{body}^{-1}\mathbf{R}^{\mathrm{T}} \\ \tfrac{d}{dt}\mathbf{I}^{-1} &= (\tfrac{d}{dt}\mathbf{R})\mathbf{I}_{body}^{-1}\mathbf{R}^{\mathrm{T}} + \mathbf{R}\mathbf{I}_{body}^{-1}(\tfrac{d}{dt}\mathbf{R}^{\mathrm{T}}) \\ &= \omega^*\mathbf{R}\mathbf{I}_{body}^{-1}\mathbf{R}^{\mathrm{T}} + \mathbf{R}\mathbf{I}_{body}^{-1}\mathbf{R}^{\mathrm{T}}\omega^{*\mathrm{T}} \\ &= \omega^*\mathbf{I}^{-1} + \mathbf{I}^{-1}\omega^{*\mathrm{T}} \end{aligned} \tag{B.8}$$

We have substituted $\omega^*\mathbf{R}$ for $\frac{d}{dt}\mathbf{R}$, according to the equations of motion (Appendix A).

We will also need the derivative of $\vec{\omega}$:

$$\begin{aligned} \vec{\omega} &= \mathbf{I}^{-1}\vec{L} \\ \tfrac{d}{dt}\vec{\omega} &= (\tfrac{d}{dt}\mathbf{I}^{-1})\vec{L} + \mathbf{I}^{-1}(\tfrac{d}{dt}\vec{L}) \\ &= (\tfrac{d}{dt}\mathbf{I}^{-1})\vec{L} + \mathbf{I}^{-1}\vec{T} \\ &= \omega^*\mathbf{I}^{-1}\vec{L} + \mathbf{I}^{-1}\omega^{*\mathrm{T}}\vec{L} + \mathbf{I}^{-1}\vec{T} \\ &= \omega^*\vec{\omega} + \mathbf{I}^{-1}(\omega^{*\mathrm{T}}\vec{L} + \vec{T}) \\ &= \mathbf{I}^{-1}(\vec{L} \times \vec{\omega} + \vec{T}) \end{aligned} \tag{B.9}$$

We have substituted $\vec{T}$ for $\frac{d}{dt}\vec{L}$ according to the equations of motion.

We can now differentiate $\vec{b}$:

$$\begin{aligned} \vec{b} &= \mathbf{R}\vec{b}_{body}, \text{ and} \\ \tfrac{d}{dt}\vec{b} &= \tfrac{d}{dt}(\mathbf{R}\vec{b}_{body}) \\ &= (\tfrac{d}{dt}\mathbf{R})\vec{b}_{body} \\ &= \omega^*\mathbf{R}\vec{b}_{body} \\ &= \omega^*\vec{b} \\ &= \vec{\omega} \times \vec{b} \\ \tfrac{d^2}{dt^2}\vec{b} &= (\tfrac{d}{dt}\vec{\omega}) \times \vec{b} + \vec{\omega} \times (\tfrac{d}{dt}\vec{b}) \\ &= (\tfrac{d}{dt}\vec{\omega}) \times \vec{b} + \vec{\omega} \times (\vec{\omega} \times \vec{b}) \\ &= (\mathbf{I}^{-1}(\vec{L} \times \vec{\omega} + \vec{T})) \times \vec{b} + \vec{\omega} \times (\vec{\omega} \times \vec{b}) \\ &= (\mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1})\vec{T} + (\mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1}(\vec{L} \times \vec{\omega}) + \vec{\omega} \times (\vec{\omega} \times \vec{b})) \\ &= \mathbf{\Lambda}\vec{T} + \vec{\beta} \end{aligned}$$

where we define

$$\begin{aligned} \mathbf{\Lambda} &= \mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1} \\ \vec{\beta} &= \mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1}(\vec{L} \times \vec{\omega}) + \vec{\omega} \times (\vec{\omega} \times \vec{b}) \end{aligned} \tag{B.10}$$

We have again substituted $\omega^*\mathbf{R}$ for $\frac{d}{dt}\mathbf{R}$.

Finally, we can express the position, velocity, and acceleration of point P in terms of the state of the body and the net force and torque on the body:

$$\begin{aligned} \vec{X}_P &= \vec{x} + \vec{b} \\ \vec{v}_P &= \tfrac{d}{dt}\vec{X}_P \\ &= \tfrac{d}{dt}\vec{x} + \tfrac{d}{dt}\vec{b} \\ &= \vec{v} + \vec{\omega} \times \vec{b} \\ \vec{a}_P &= \tfrac{d^2}{dt^2}\vec{X}_P \\ &= \tfrac{d^2}{dt^2}\vec{x} + \tfrac{d^2}{dt^2}\vec{b} \\ &= \tfrac{1}{m}\vec{F} + \mathbf{\Lambda}\vec{T} + \vec{\beta} \\ &= \mathbf{\Gamma}\vec{F} + \mathbf{\Lambda}\vec{T} + \vec{\beta} \end{aligned} \tag{B.11}$$

where we define

$$\mathbf{\Gamma} = \tfrac{1}{m}$$

Appendix C

Constraint Derivations

C.1 Adding New Constraints

As expected, it has been quite easy to add new types of constraints to the modeling system of Ch. 5 Typically, starting with a twice-differentiable deviation function $\vec{D}(\mathcal{Y}_s, t)$, we differentiate and "type in" the resulting equations, as described in Appendix C.2. One must be aware, however, of the "gotchas" that are described in Sec. 4.2.3 and Sec. 4.2.4.

There is some freedom in choosing the forces and torques that solve the constraint equation. In general, an arbitrary force and torque at the center of mass suffices, giving us six degrees of freedom; thus we would have the following definitions:

$$\vec{F}_c = \begin{bmatrix} F_x \\ F_y \\ F_z \\ T_x \\ T_y \\ T_z \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad \text{(C.1)}$$

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

If the force and torque are applied at a point P (not the center of mass) at radius vector $\vec{b}$ from the center of mass (see Fig. A.2), then the six degrees of freedom translate to force and torque at the center of mass via:

$$\vec{F}_c = \begin{bmatrix} F_x \\ F_y \\ F_z \\ T_x \\ T_y \\ T_z \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \qquad \text{(C.2)}$$

$$\mathbf{H} = \begin{bmatrix} 0 & b_z & -b_y & 1 & 0 & 0 \\ -b_z & 0 & b_x & 0 & 1 & 0 \\ b_y & -b_x & 0 & 0 & 0 & 1 \end{bmatrix}$$

For some types of constraints, only a constraint force or only a constraint torque is needed to solve the constraint equation; thus three degrees of freedom are needed instead of six. Since solving the constraint-force equation typically takes $O(n^3)$ operations, this can significantly reduce the computation time. If the three degrees of freedom correspond to a force, Eqn. C.2 becomes

$$\vec{F}_c = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = 1 \qquad \text{(C.3)}$$

$$\mathbf{H} = \begin{bmatrix} 0 & b_z & -b_y \\ -b_z & 0 & b_x \\ b_y & -b_x & 0 \end{bmatrix} = \mathbf{b}^*$$

The constraints in Appendix C.3 each have three degrees of freedom; the "point-to-nail," "point-to-path," and "point-to-point" constraints have **G** and **H** in the above form.

When a single constraint involves several bodies, we generally wish to meet the constraint without introducing a net change in momentum. The easiest way to achieve this is to introduce equal and opposite pairs of forces. The "point-to-point" constraint (Appendix C.3.3) does this.

C.2 Outline of Derivation Process

For each type of constraint, we must derive expressions for the various quantities defined in Fig. 3.6. The steps we follow are:

1. Choose a simple "deviation" measure $\vec{D}$. $\vec{D}$ is a function of the positions ($\vec{x}$) and orientations ($\mathbf{R}$) of the constrained bodies, and may optionally depend on t.

2. Differentiate $\vec{D}$, to derive $\vec{D}^{(1)}(\mathcal{Y}, t)$, Substituting $\vec{v}$ and $\omega^* R$ for $\frac{d}{dt}\vec{x}$ and $\frac{d}{dt}R$ terms which will arise (see Fig. 3.6).

3. Differentiate again, to derive $\vec{D}^{(2)}(\mathcal{Y}, \vec{F}, \vec{T}, t)$. Replace $\frac{d}{dt}\vec{p}$ and $\frac{d}{dt}\vec{L}$ terms with $\vec{F}$ and $\vec{T}$, thus giving rise to the linear dependence of $\vec{D}^{(2)}$ on the forces and torques. Define the $d \times 3$ matrices $\mathbf{\Gamma}$, $\mathbf{\Lambda}$, and the d-vector $\vec{\beta}$

4. Decide how to apply constraint forces to meet this constraint. Most often, we apply an arbitrary vector force to some fixed location of the constrained body; in this case, we have $f = 3$ degrees of freedom.

5. Write down $\mathbf{G}$ and $\mathbf{H}$ for each body in the constraint. These convert the f values in the "constraint force" $\vec{F}_c$ into the actual forces and torques on the bodies.

Often, some of the quantities $\mathbf{\Gamma}$, $\mathbf{\Lambda}$, $\mathbf{G}$, and $\mathbf{H}$, which are nominally matrices, turn out to be scalar. Scalars can be handled as a special case in the implementation, or scaled identity matrices can be used.

We give examples of the constraint derivations for the constraints illustrated in Ch. 1.

C.3 Examples

C.3.1 "Point-to-Nail" Constraint

We choose $\vec{D}$ to be the vector from the constrained point, at $\vec{X}_P$, to the "nail", at $\vec{X}_0$ (see Fig. 3.2). We thus have:

$$\begin{aligned} d &= 3 \\ \vec{D}(\mathcal{Y}) &= \vec{X}_P - \vec{X}_0 \end{aligned}$$

The differentiation in Appendix B immediately gives us the quantities:

$$\begin{aligned} \vec{D}(\mathcal{Y}) &= \vec{X}_P(\mathcal{Y}) - \vec{X}_0 \\ \vec{D}^{(1)}(\mathcal{Y}) &= \vec{v}_P(\mathcal{Y}) \\ \vec{D}^{(2)}(\mathcal{Y}, \vec{F}, \vec{T}) &= \vec{a}_P(\mathcal{Y}, \vec{F}, \vec{T}) \\ \mathbf{\Gamma} &= \frac{1}{m} \\ \mathbf{\Lambda} &= \mathbf{b}^{*T}\mathbf{I}^{-1} \\ \vec{\beta} &= \mathbf{b}^{*T}\mathbf{I}^{-1}(\vec{L} \times \vec{\omega}) + \vec{\omega} \times (\vec{\omega} \times \vec{b}) \end{aligned}$$

We apply an arbitrary force to the constrained point. The "constraint force" $\vec{F}_c$ is used directly as a force on the body, and it contributes $\vec{b} \times \vec{F}_c$ to the torque.

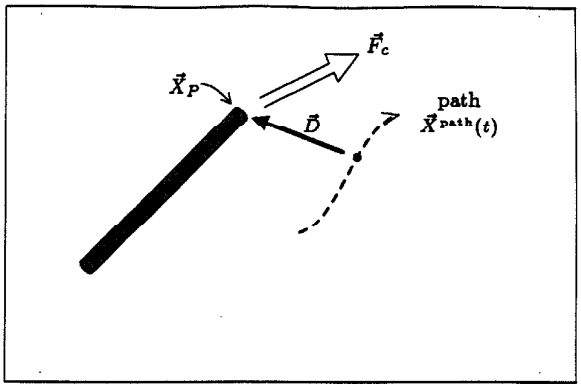


Figure C.1: "Point-to-path" Constraint. $\vec{D}(\mathcal{Y}, t) = \vec{X}_P(\mathcal{Y}) - \vec{X}^{\text{path}}(t)$

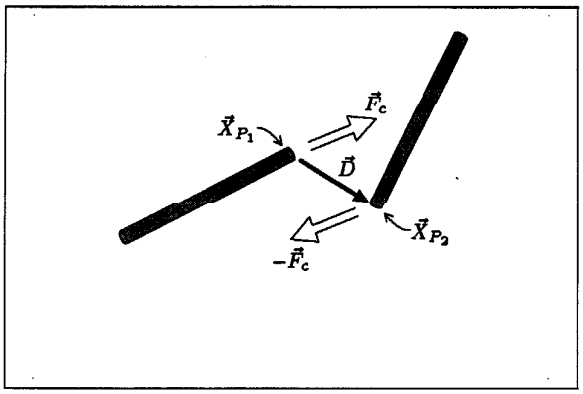


Figure C.2: "Point-to-point" constraint. $\vec{D}(\mathcal{Y}) = \vec{X}_{P_2}(\mathcal{Y}) - \vec{X}_{P_1}(\mathcal{Y})$

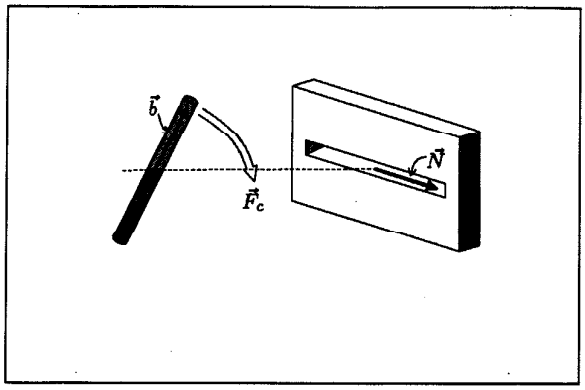


Figure C.3: "Orientation" constraint. $\vec{D}(\mathcal{Y}) = \vec{b} \cdot \vec{N} - 1$. Constraint torque is $\vec{F}_c$; There is no constraint force.

We thus have:

$$\begin{array}{ll} f & = 3 \\ \mathbf{G} & = 1 \\ \mathbf{H} & = \mathbf{b}^* \end{array}$$

C.3.2 "Point-To-Path" Constraint

This constraint is met at a time t if the constrained point "P" is on the path, at "$\vec{X}^{\text{path}}(t)$" (Fig. C.1); it is the same as the "point-to-nail" constraint, but with a nail that moves. Thus when we differentiate, we keep terms which depend on $\vec{X}^{\text{path}}$, but otherwise the derivation is the same as for the "point-to-nail." We have:

$$\begin{array}{ll} d & = 3 \\ \vec{D}(\mathcal{Y},t) & = \vec{X}_P(\mathcal{Y}) - \vec{X}^{\text{path}}(t) \\ \vec{D}^{(1)}(\mathcal{Y},t) & = \vec{v}_P(\mathcal{Y}) - \frac{d}{dt}\vec{X}^{\text{path}}(t) \\ \vec{D}^{(2)}(\mathcal{Y},\vec{F},\vec{T},t) & = \vec{a}_P(\mathcal{Y},\vec{F},\vec{T}) - \frac{d^2}{dt^2}\vec{X}^{\text{path}}(t) \\ \mathbf{\Gamma} & = \frac{1}{m} \\ \mathbf{\Lambda} & = \mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1} \\ \vec{\beta} & = \mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1}(\vec{L} \times \vec{\omega}) + \vec{\omega} \times (\vec{\omega} \times \vec{b}) \\ & \quad - \frac{d^2}{dt^2}\vec{X}^{\text{path}}(t) \\ f & = 3 \\ \mathbf{G} & = 1 \\ \mathbf{H} & = \mathbf{b}^* \end{array}$$

Notice the explicit dependence of $\vec{D}$, $\vec{D}^{(1)}$, and $\vec{\beta}$ on t.

C.3.3 "Point-To-Point" Constraint

This constraint is met if the two constrained points "P_1" and "P_2" are at the same location (Fig. C.2). We thus define $\vec{D}$ to be the vector separating the two points. The derivation proceeds analogously to that of the "point-to-nail" constraint:

$$\begin{array}{ll} d & = 3 \\ \vec{D}(\mathcal{Y}) & = \vec{X}_{P_2}(\mathcal{Y}) - \vec{X}_{P_1}(\mathcal{Y}) \\ \vec{D}^{(1)}(\mathcal{Y}) & = \vec{v}_{P_2}(\mathcal{Y}) - \vec{v}_{P_1}(\mathcal{Y}) \\ \vec{D}^{(2)}(\mathcal{Y},\vec{F},\vec{T}) & = \vec{a}_{P_2}(\mathcal{Y}) - \vec{a}P_1(\mathcal{Y}) \\ \mathbf{\Gamma}^1 & = -\frac{1}{m_1}\mathbf{1} \\ \mathbf{\Lambda}^1 & = -\mathbf{b}_1^{*\mathrm{T}}\mathbf{I}_1^{-1} \\ \mathbf{\Gamma}^2 & = \frac{1}{m_2}\mathbf{1} \\ \mathbf{\Lambda}^2 & = \mathbf{b}_2^{*\mathrm{T}}\mathbf{I}_2^{-1} \end{array}$$

To be in keeping with Newton's third law, the two bodies must exert equal and opposite forces on each other. We apply an arbitrary force, $\vec{F}_c$, to one of the constrained points, and the negation of that force, $-\vec{F}_c$, to the other. We thus have:

$$\begin{array}{ll} f & = 3 \\ \mathbf{G}^1 & = -1 \\ \mathbf{H}^1 & = -\mathbf{b}_1^* \\ \mathbf{G}^2 & = 1 \\ \mathbf{H}^2 & = \mathbf{b}_2^* \end{array}$$

C.3.4 "Orientation" Constraint

This constraint is met if a specified unit vector $\vec{b}$ fixed in the body lines up with a unit vector $\vec{N}$ fixed in the world (see Fig. C.3). We could define D to be the angle between the vectors; it easier, however, if we define D to be 0 when the cosine of the angle (i.e. the dot-product of the vectors) is 1. Thus we have:

$$\begin{array}{lcl} d & = & 1 \\ D(\mathcal{Y}) & = & \vec{b}(Y) \cdot \vec{N} - 1 \\ D^{(1)}(\mathcal{Y}) & = & (\frac{d}{dt}\vec{b}(\mathcal{Y})) \cdot \vec{N} \\ D^{(2)}(\mathcal{Y},\vec{T}) & = & (\frac{d^2}{dt^2}\vec{b}(\mathcal{Y})) \cdot \vec{N} \\ & = & ((\mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1})\vec{T}) \cdot \vec{N} \\ & & +(\mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1}(\vec{L} \times \vec{\omega}) + \vec{\omega} \times (\vec{\omega} \times \vec{b})) \cdot \vec{N} \\ \mathbf{\Gamma} & = & 0 \\ \mathbf{\Lambda} & = & \vec{N}^{\mathrm{T}}\mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1} \\ \vec{\beta} & = & (\mathbf{b}^{*\mathrm{T}}\mathbf{I}^{-1}(\vec{L} \times \vec{\omega}) + \vec{\omega} \times (\vec{\omega} \times \vec{b})) \cdot \vec{N} \end{array}$$

Notice that $\mathbf{\Lambda}$ is a 1×3 matrix, and $\vec{\beta}$ is a scalar.

We apply an arbitrary pure torque, $\vec{F}_c$, to the body, and no force. We therefore have:

$$\begin{array}{ll} f & = 3 \\ \mathbf{G} & = 0 \\ \mathbf{H} & = 1 \end{array}$$

Notice that this constraint is "non-square" – we are applying 3 degrees of freedom to affect a scalar constraint "deviation."

Appendix D

Object Collision

We would like to incorporate collisions between bodies into our physical model. Bodies should not pass through each other, ghost-like; instead, we would like to detect collisions and cause the objects to interact, e.g. by bouncing or sticking.

We will give a brief discussion of the basic issues involved with object collision, along with brief descriptions of some work we have done. A full discussion of collisions and collision-handling is beyond the scope of this thesis.

D.1 Detecting Collisions

The problem of detecting intersections in a collection of arbitrary bodies is as yet an open research question. A recent work that addresses this problem is [Von Herzen 88].

For our work, we focus on the question of how to handle a collision, given that it has been detected. We assume a "black box" that reports collisions and provides the necessary data such as time and point of collision. In the work we have done so far, we have used simple geometric bodies, and have merely tested every body against every other.

D.2 Handling Collisions

Fig. D.1 examines what happens when a body collides with a wall, and ricochets. If we model the ricochet as an instantaneous change in velocity, we see that the force on the object must be infinite (a delta function).

D.2.1 Via Constraints

It is appealing to write a non-interpenetration constraint, that is zero when a pair of bodies do not intersect, and is non-zero when they do. The constraint described in Eqn. 4.4 is an example of such a constraint. Unfortunately, as discussed in Sec. 4.2.4, such a constraint can cause great difficulty for the numerical differential-equation solver, as it tries to integrate the discontinuous force.

D.3 Via Boundary Layers

Another choice is to create a small (but non-zero) boundary layer around each object; when an object penetrates the boundary layer of another object, a constraint can be turned on to keep the object from penetrating. One can trade off accuracy of the model vs. stiffness of the differential equations by adjusting the size of the boundary layer. Appendix F.5 shows frames from an animation of a donut dropping onto a table; the boundary-layer approach was used to keep the donut from penetrating the table.

D.4 Via Impulses

The most appealing way to handle collisions is using the techniques of classical mechanics to calculate the transfer of momentum between bodies via impulses. This can incorporate such parameters as the elasticity of the collision.

In following such an approach, we must recognize that the O.D.E. $\frac{d}{dt}\mathcal{Y} = f(\mathcal{Y}, t)$ that is presented in Appendix A does not completely describe the dynamics of the model. Instead, the equation we must solve is a *piecewise* O.D.E.: It is an O.D.E. until a collision, at which point the state $\mathcal{Y}$ is changed instantaneously in accordance with the impulse calculation, and the result is used as the initial conditions of a new O.D.E.

We have successfully built a piecewise-O.D.E. solver, and have used it, along with the classical impulse equations, to simulate collisions between constrained bodies. [Constrained Dynamics 88] contains some animations using this technique.

When applying impulses to the model, one must be careful not to "break" existing constraints. The constrained impulse calculation turns out to be a

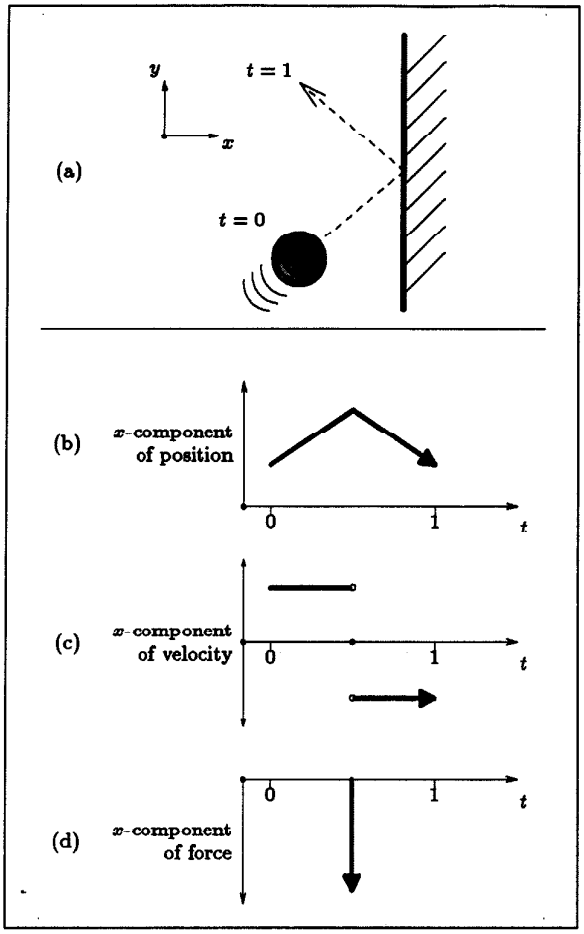


Figure D.1: Collisions. If an objects ricochets off another object, there is a discontinuity in velocity. In attempting to achieve this via a differential equation solver, the force becomes very large, and the solver's step size become very small. (a) A ball bounces off a wall; (b) Graph of horizontal position vs. time (c) Graph of horizontal velocity vs. time (d) horizontal force on the ball vs. time is a delta function.

variant of the constraint-force equation $\mathcal{M}\mathcal{F}_c+\mathcal{B}=0$; a full discussion is beyond the scope of this paper.

Appendix E

Isomers

When specifying objects by connectivity constraints, it is often possible to build "isomers" (see Fig. E.1). Which isomer gets built depends on the initial conditions, the time constants, and the external forces on the bodies.

If there is more than one configuration of the body which satisfies the constraints, we will generally produce isomers. If the model requires a particular isomer, it is best to to disambiguate the solution by specifying further constraints.

Currently, there are several ad-hoc techniques which can be used to attempt to coerce the system into assembling the desired isomer: The modeler can adjust the initial placement of the bodies, or activate the constraints sequentially, or add "guiding" non-constraint forces. The tower-assembly sequence (Appendix F.3) made use of these techniques. These techniques are not guaranteed to work, however, and may in fact lead the modeler down "garden paths."

We are examining techniques for meeting constraints involving inequalities; such constraints would greatly aid in disambiguating isomers. An example might be "make sure that object A is above object B".

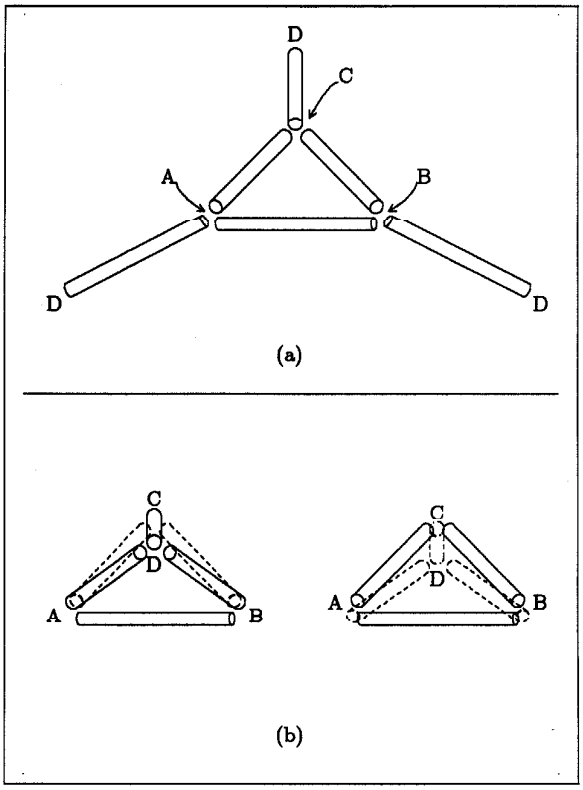


Figure E.1: Alternate "isomeric" solutions to a single set of constraints. (a) The points labeled "D" are to be connected together via "point-to-point" constraints, as are the points labeled "A", "B", and "C".' (b) Two different tetrahedra meet the constraints: the "D" points can be connected above or below the other points.

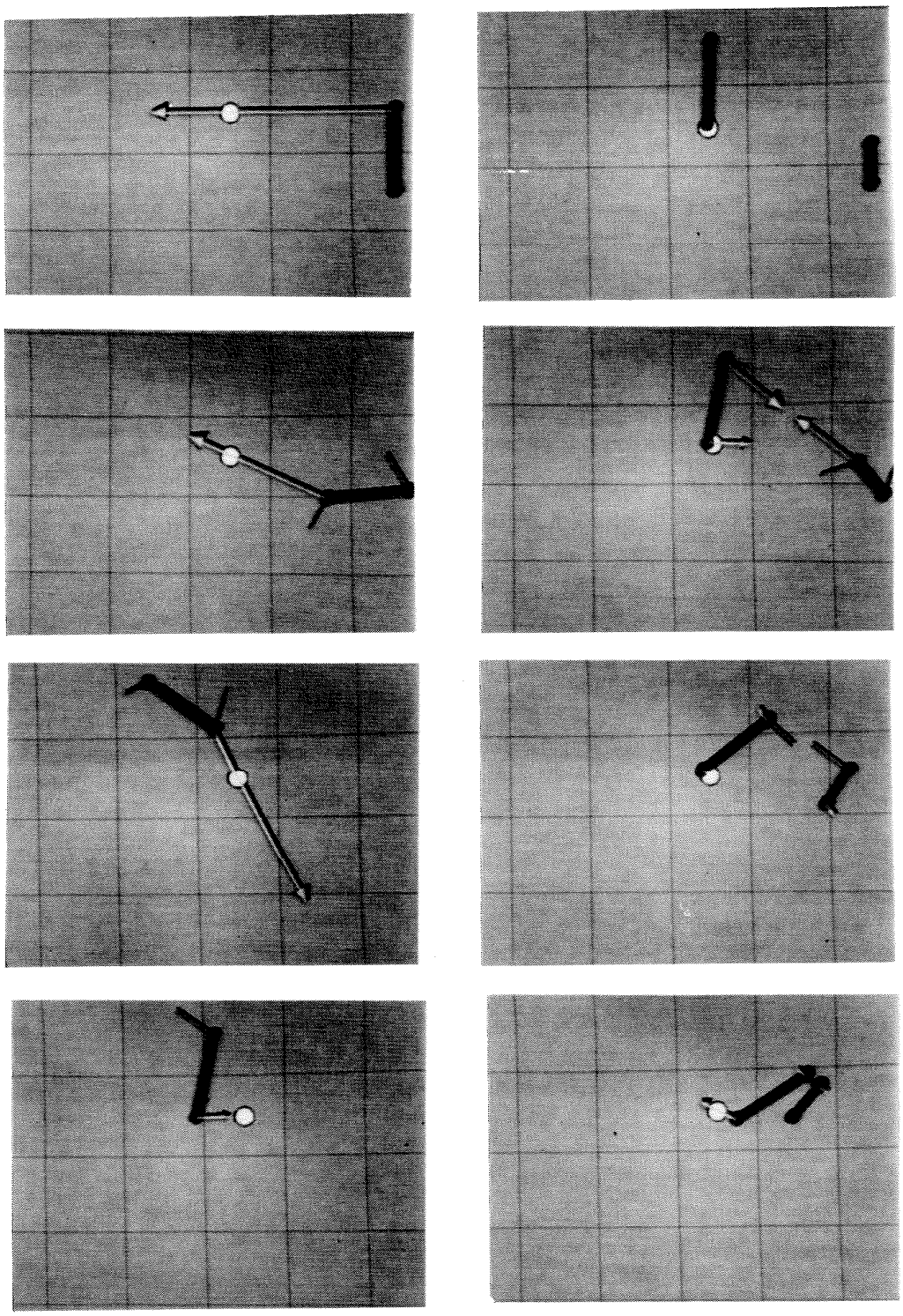
Appendix F

Animation Sequences

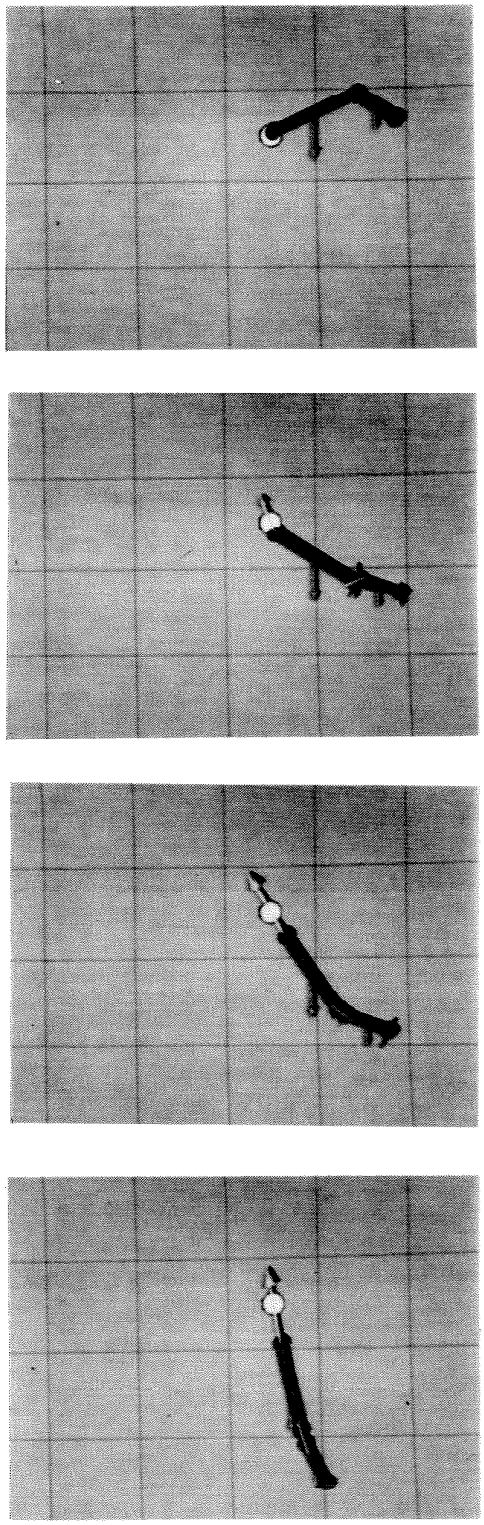
This appendix contains frames from various modeling and animation sequences that have been made using the Dynamic Constraints modeling system. Most of the sequences appear on the videotape [Caltech '87 Demo Reel].

F.1 "Rubber Bands"

These frames illustrate attempts to use "rubber bands" to make a compound pendulum, as in Fig. 1.7. The use of "rubber bands" is discussed in Sec. 2.2. Compare with the "Dynamic Constraints" sequence in Appendix F.2. *Left:* A damped "rubber band" meets a "point-to-nail" constraint. *Right:* We connect a second stick to the first; notice that the first point is temporarily pulled away from the nail.

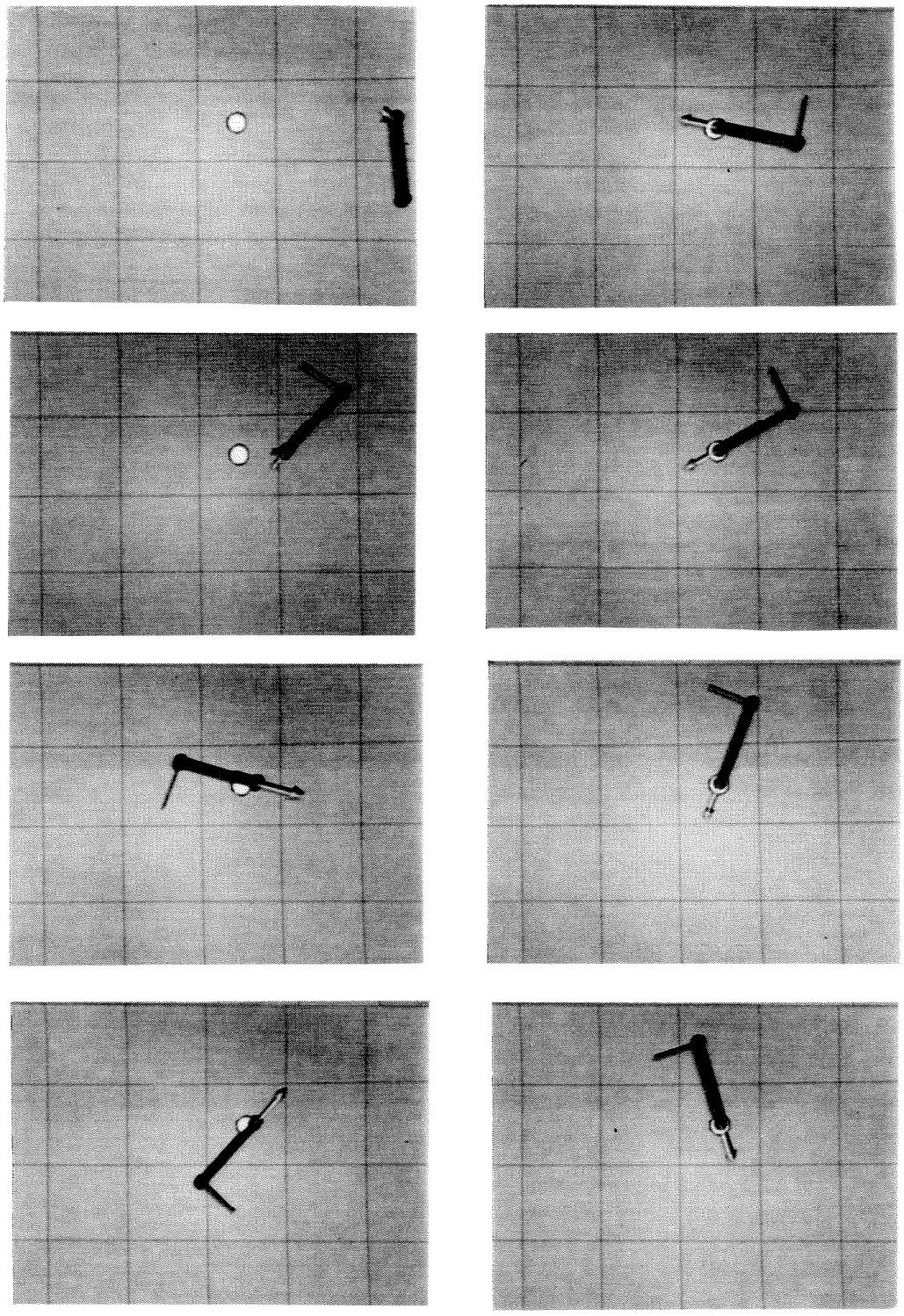


"Rubber bands," continued. We apply an external "gravity" force to the sticks. Notice that the rubber bands cannot keep the constraints met; the sticks pull apart.

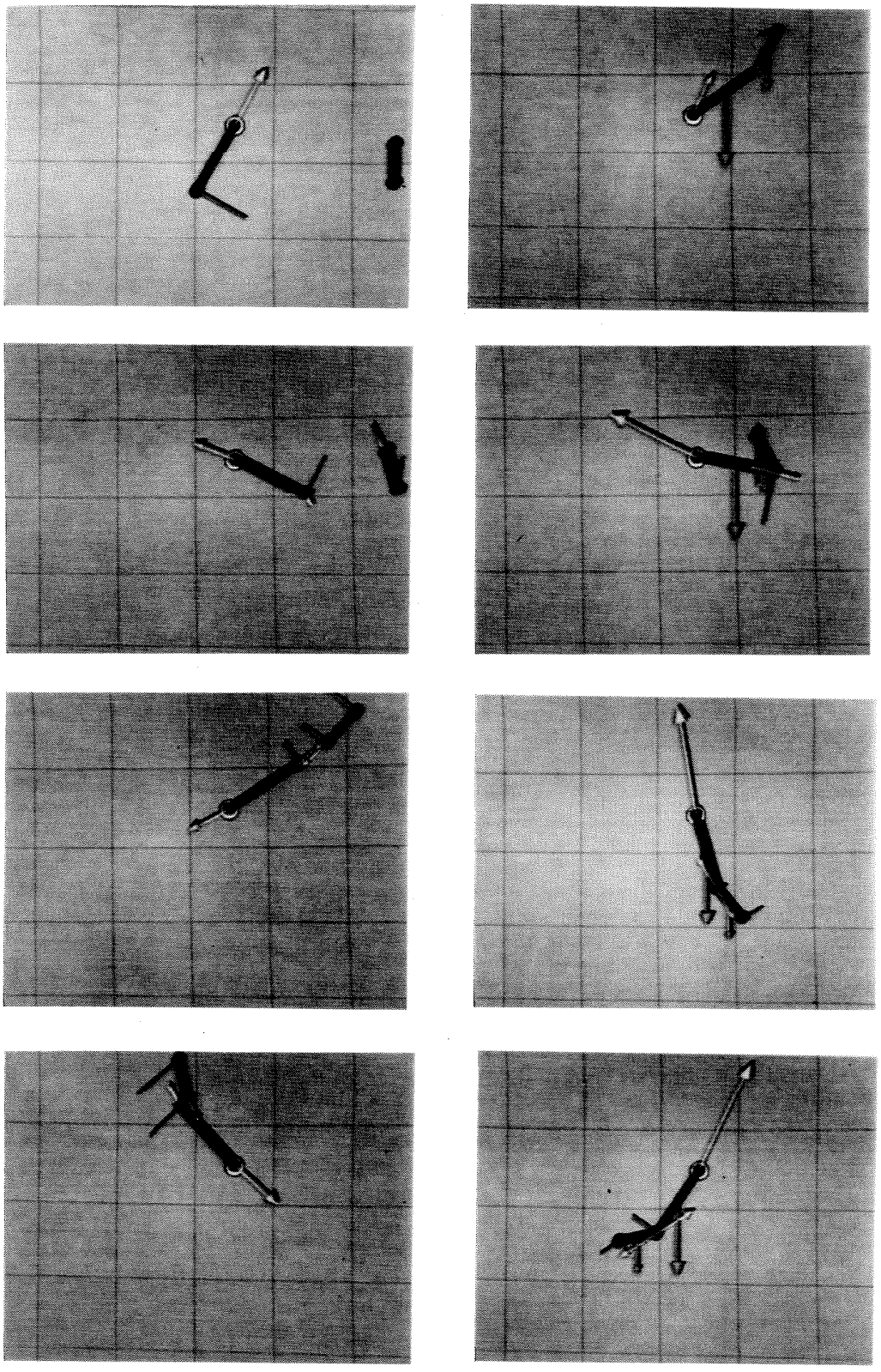


F.2 Dynamic Constraints

These frames illustrate "Dynamic Constraints" being used to make a compound pendulum, as in Fig. 1.7. Compare with the "Rubber Bands" sequence in Appendix F.1. *Left:* The constraint force pulls the end of the stick in a straight line towards the nail. *Right:* Once the point is at the nail, the stick revolves freely; note that the constraint force is radial, providing the necessary radial acceleration allowing the stick to revolve.

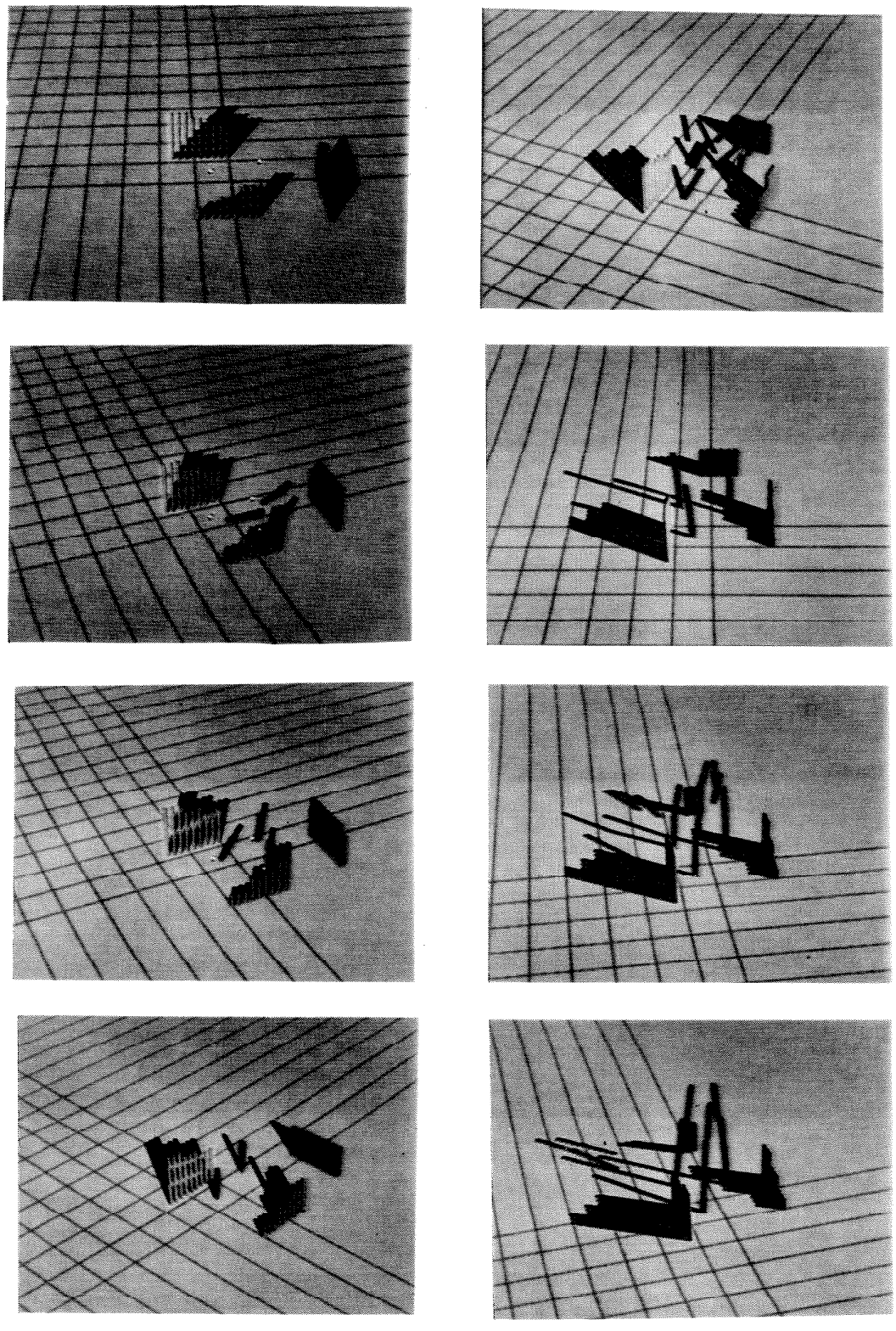


Dynamic Constraints, continued. *Left:* A second stick is attached to the first; notice that the first stick stays attached to the nail while the second stick joins it. *Right:* An external "gravity" force is applied to the sticks. The constraint forces adapt to hold the compound pendulum.

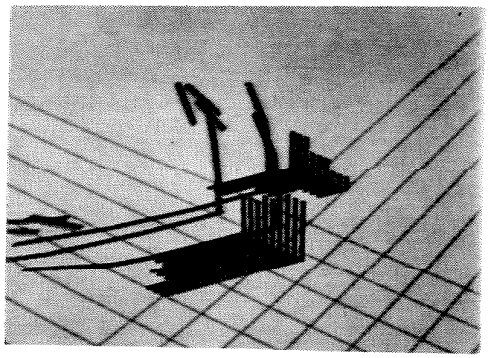
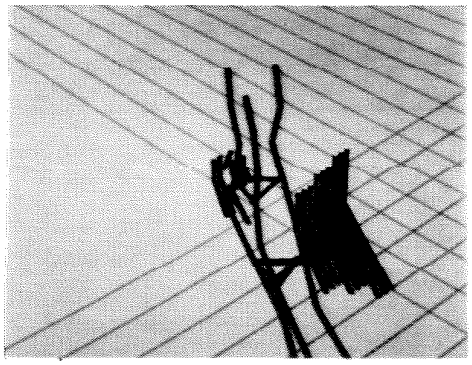
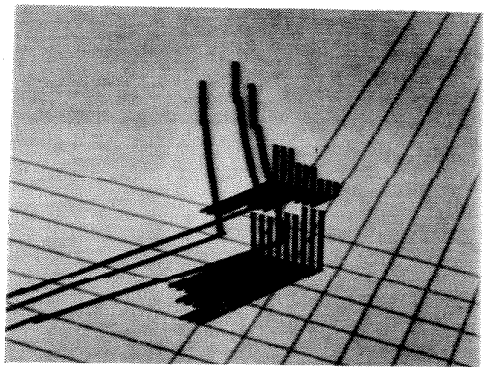
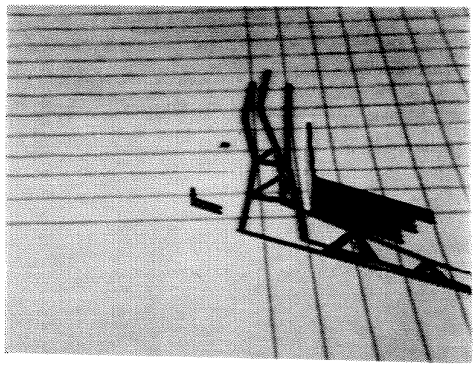
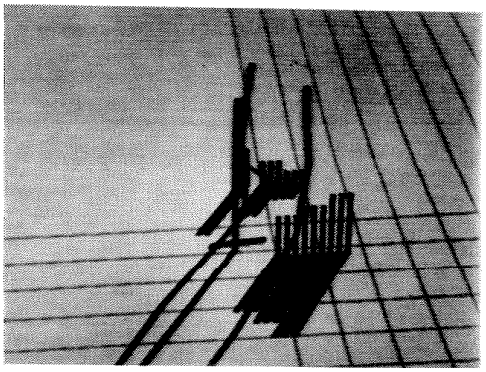
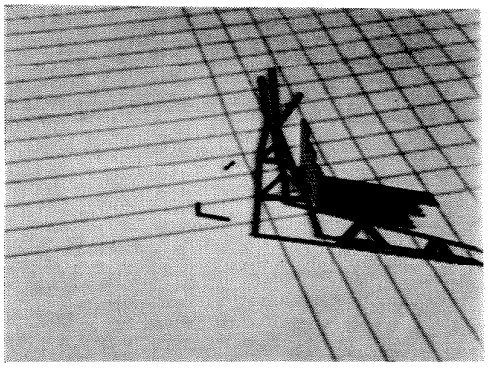
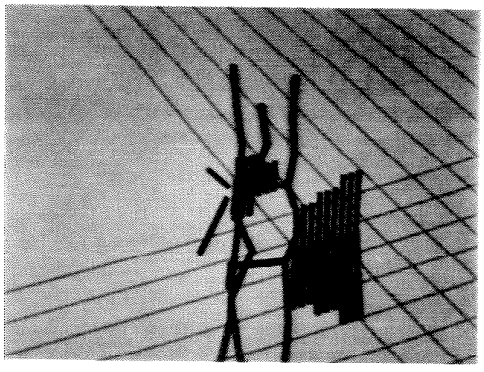
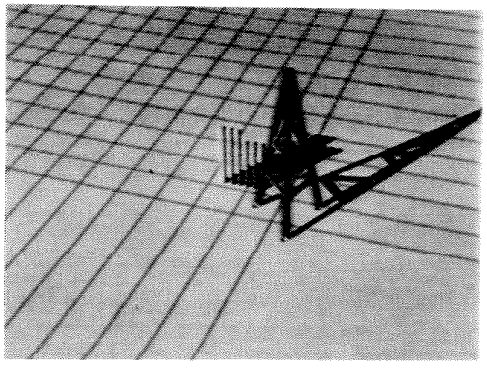


F.3 Tower Assembly

Frames from an animation showing the self-assembly of a tower using "point-to-point" and "point-to-nail" constraints. A buoyant force is applied to the legs of the tower, to keep them upright until the constraints are met.

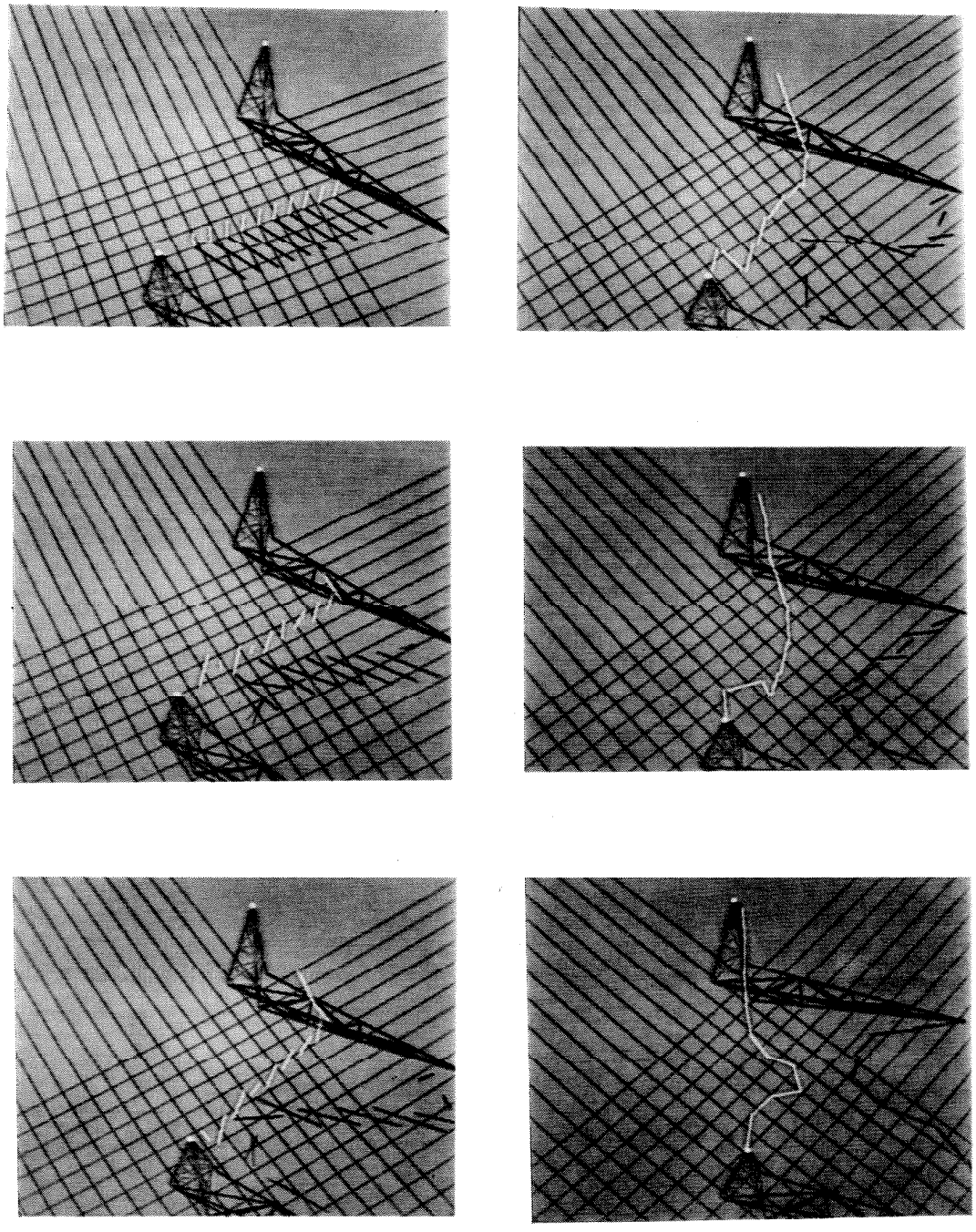


Tower Assembly, continued.

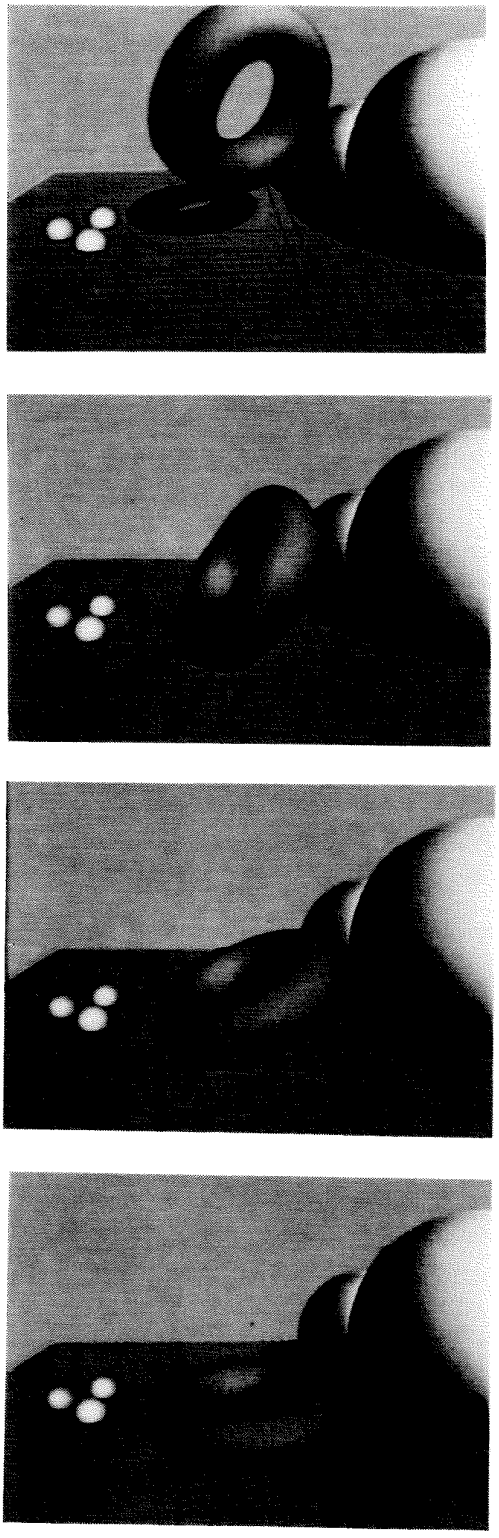
F.4 Linked Chain

Frames from an animation showing the assembly and subsequent motion of a chain between two towers. The links are connected via "point-to-point" constraints. Gravity and damping are applied to the links. *Left:* The links assemble to form the chain. *Right:* The chain jangles.



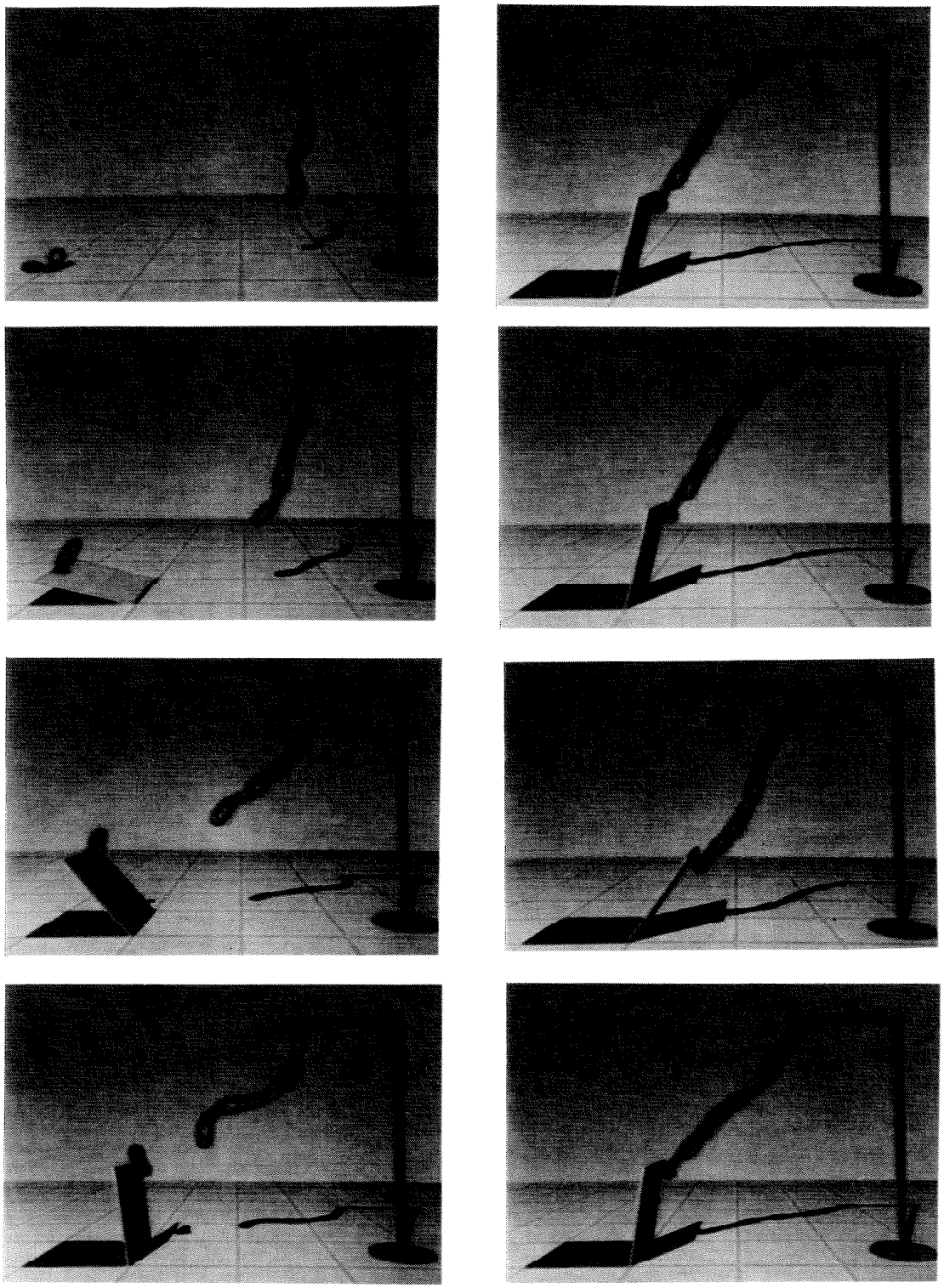
F.5 Donut Falling onto Table

Frames from an animation testing non-interpenetration constraints; the donut is not allowed to pass through the table. See discussion in Appendix D.



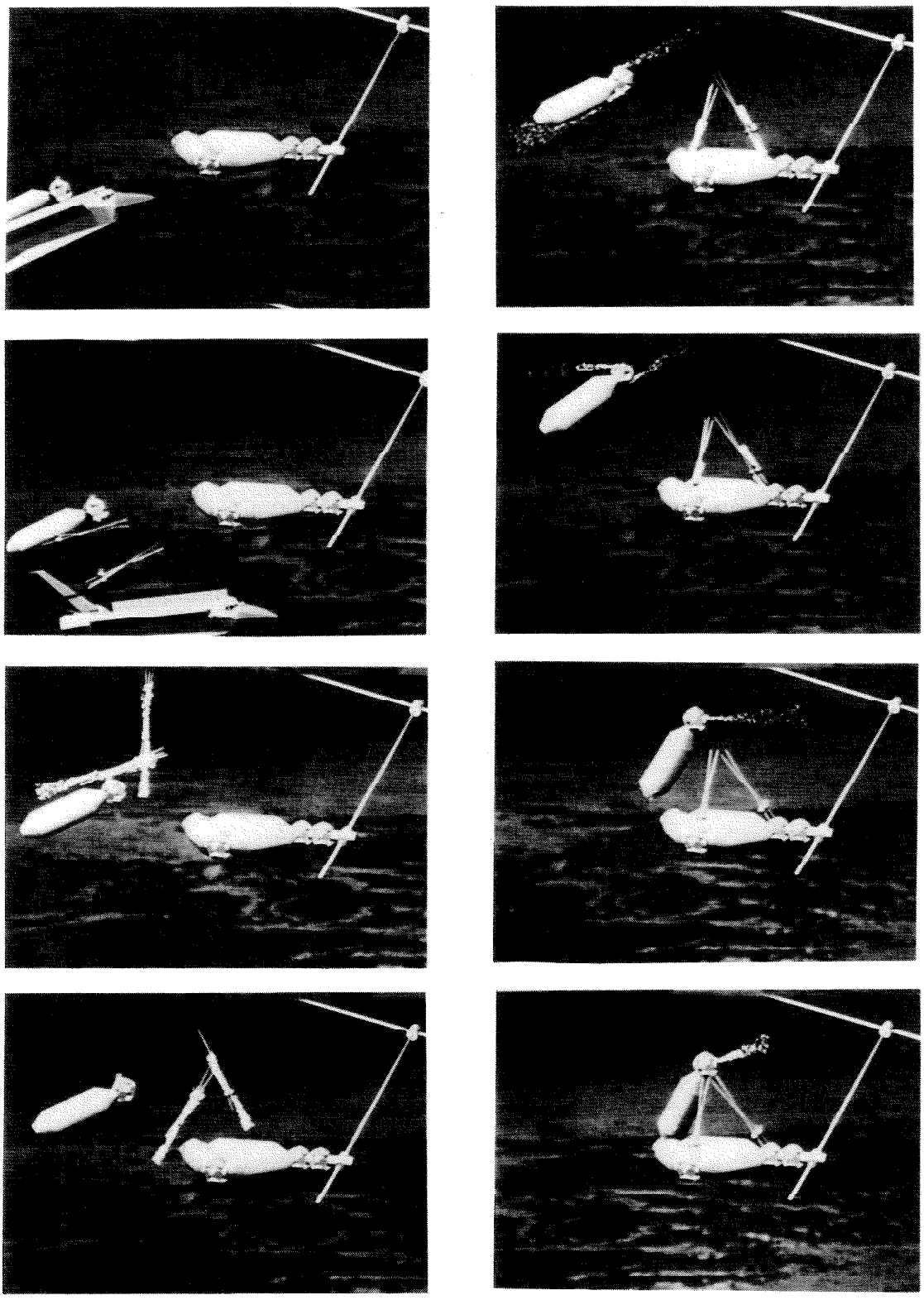
F.6 Pandora's Chain

Frames from an animation of a lively chain. The chain is assembled via "point-to-point" and "point-to-nail" constraints. Torsion springs (see glossary) are placed between the links, to keep the links roughly perpendicular, and to add liveliness. *Left:* The chain attaches to the hook on the trap door, lifting the door. *Right:* The resulting system swings.



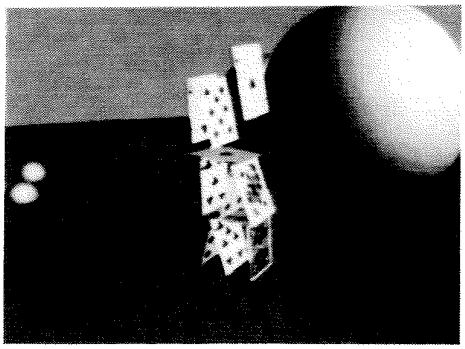
F.7 Space Station Assembly

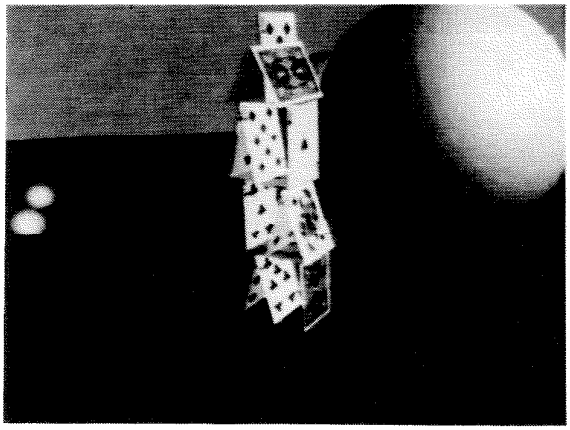
Frames from an animation showing self-assembly of a modular space station. The modules are assembled via "orientation," "point-to-point," and "point-to-nail" constraints. The constraint forces that are computed determine the strengths of the rocket thrusts.



F.8 Card House

A cardhouse is assembled using "point-to-point" and "point-to-plane" constraints.





Bibliography

[Armstrong and Green 85] Armstrong, William W., and Mark W. Green, *The dynamics of articulated rigid bodies for purposes of animation,* in **Visual Computer,** Springer-Verlag, 1985, pp. 231-240.

[Barr 83] Barr, Alan H., *Geometric Modeling and Fluid Dynamic Analysis of Swimming Spermatozoa,* Ph.D. Dissertation, Rensselaer Polytechnic Institute, 1983

[Barr, Von Herzen, Barzel, and Snyder 87] Barr, Alan H., Brian Von Herzen, Ronen Barzel, and John Snyder, *Computational Techniques for the Self Assembly of Large Space Structures* Proceedings of the 8th Princeton/AIAA/SSI Conference, American Institute of Aeronautics and Astronautics, 1987, pp.275-282

[Barzel and Barr 88] Barzel, Ronen, and Alan H. Barr, *A Modeling System Based on Dynamic Constraints,* Proc. SIGGRAPH, 1988.

[Boyce and Deprima 77] Boyce, William E., and DiPrima, Richard C., **Elementary Differential Equations and Boundary Value Problems,** John Wiley & Sons, New York, 1977.

[Caltech '87 Demo Reel] **Caltech studies in modeling and motion** (videotape), in SIGGRAPH video Review #28, *Visualization in Scientific Computing Computer Graphics,* volume 21 number 6. ACM SIGGRAPH, 1987

[Constrained Dynamics 88] **Constrained Dynamics** (videotape), Caltech Computer Graphics Group, 1988

[Fox 67] Fox, E.A., **Mechanics,** Harper and Row, New York, 1967

[Galiullin 84] Galiullin, A.S., **Inverse Problems of Dynamics,** Mir Publishers, Moscow, 1984

[Gear 71] Gear, C. William, **Numerical Initial Value Problems in Ordinary Differential Equations,** Prentice-Hall, Englewood Cliffs, NJ, 1971

[Gerard and Maciejewski 85] Gerard, Michael, and A.A. Maciejewski, *Computational Modeling for the Computer Animation of Legged Figures,* Computer Graphics, Vol. 19 No. 3, July 1985, pp. 263-270.

[Goldstein 80] Goldstein, Herbert, **Classical Mechanics,** Second Edition, Addison-Wesley, Reading, Massachusetts, 1980.

[Golub and Van Loan 83] Golub, G., and Van Loan, C., **Matrix Computations,** Johns Hopkins University Press, Baltimore, 1983.

[Isaacs and Cohen 87] Isaacs, Paul M. and Michael F. Cohen, *Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions, and Inverse Dynamics,* Proc. SIGGRAPH 1987, pp. 215-224

[Lengyel 87] Lengyel, Jed, *Dynamic Assembly and Behavioral Simulation of the Flagellar Axoneme,* in **Caltech SURF Reports,** 1987

[Lien and Kajiya 84] Lien, Sheue-ling, and James T. Kajiya, *A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra,* IEEE Computer Graphics and Applications, Vol. 4 No. 10, Oct. 1984, pp. 35–41.

[Misner, Thorne and Wheeler 73] Misner, Charles W., Kip S. Thorne, and John Archibald Wheeler, **Gravitation,** W.H. Freeman and Co., San Francisco, 1973.

[NAG] NAG Fortran Library, Numerical Algorithms Group, 1101 31st st, Suite 100, Downers Grove, IL 60515-1263

[Press et. al. 86] Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, **Numerical Recipes/The Art of Scientific Computing,** Cambridge University Press, Cambridge, 1986.

[Platt and Barr 88] Platt, John, and Alan Barr, *Constraints on Flexible Objects,* Submitted to SIGGRAPH 1988.

[Ralston and Rabinowitz 78] Ralston, Anthony, and Philip Rabinowitz, *A First Course in Numerical Analysis,* McGraw-Hill, New York, 1978.

[Shoemake 85] Shoemake, Ken, *Animating Rotation with Quaternion Curves,* Computer Graphics, Vol. 19 No. 3, July 1985. pp. 245-254.

[Terzopoulos, Platt, Fleischer, and Barr 87] Terzopoulos, Demetri, John Platt, Alan Barr, and Kurt Fleischer *Elastically Deformable Models,* Proc. SIGGRAPH, 1987, pp. 205-214.

[Von Herzen 88] Von Herzen, Brian, *Applications of Surface Networks to Sampling Problems in Computer Graphics,* Ph.D. Dissertation, California Institute of Technology, 1988

[Witkin, Fleischer, and Barr 87] Witkin, Andrew, Kurt Fleischer, and Alan Barr, *Energy Constraints on Parametrized Models,* Proc. SIGGRAPH 1987, pp. 225-232

[Wilhelms and Barsky 85] Wilhelms, Jane, and Brian Barsky, *Using Dynamic Analysis To Animate Articulated Bodies Such As Humans and Robots,* Graphics Interface, 1985.